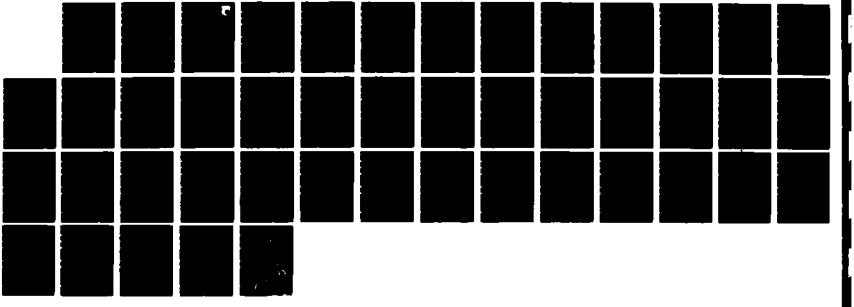
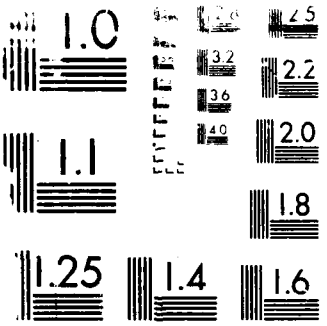
AD-A188 621 PARTS-R-US(U) CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE E BRUNVAND DEC 87 CMU-CS-87-119 AFWAL-TR-87-1170 F33615-84-K-1520 1/1

UNCLASSIFIED F/G 9/1 NL




1.0
1.1
1.25
1.4
1.6
2.0
1.8
2.2

PHOTOGRAPH THIS SHEET

AD-A188 621

DTIC ACCESSION NUMBER

LEVEL

1

INVENTORY

AFWAL-TR-87-1170

DOCUMENT IDENTIFICATION

Dec 1987

This document has been approved for public release and sale; its distribution is unlimited.

DISTRIBUTION STATEMENT

ACCESSION FOR
NTIS GRA&I
DTIC TAB
UNANNOUNCED
JUSTIFICATION

BY
DISTRIBUTION
AVAILABILITY CODES
DIST | AVAIL AND OR SPECIAL

A-1

DISTRIBUTION STAMP

DTIC ELECTE FEB 09 1988 S E D

DATE ACCESSIONED

DATE RETURNED

88 2 05 098

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NO.

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-DDAC

DTIC FORM 70A DEC 83

DOCUMENT PROCESSING SHEET

PREVIOUS EDITION MAY BE USED UNTIL STOCK IS EXHAUSTED.

AFWAL-TR-87-1170

PARTS-R-US

Erik Brunvand

Carnegie-Mellon University
Computer Science Department
Pittsburgh, PA 15213-3890

December 1987

Interim

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Chahira M. Hopper

CHAHIRA M. HOPPER
Project Engineer

Richard C. Jones

RICHARD C. JONES
Ch, Advanced Systems Research Gp
Information Processing Technology Br

FOR THE COMMANDER

Edward L. Gliatti

EDWARD L. GLIATTI
Ch, Information Processing Technology Br
Systems Avionics Div

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/AAAT , Wright-Patterson AFB, OH 45433-6543 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) CMU-CS-87-119	5 MONITORING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-87-1170
6a NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION Air Force Wright Aeronautical Laboratories AFWAL/AAAT-3
6c ADDRESS (City, State, and ZIP Code) Computer Science Dept Pittsburgh PA 15213-3890	7b ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433-6543
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-84-K-1520
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 61101E; PROJECT NO: 4976; TASK NO: 00; WORK UNIT ACCESSION NO: 01

11 TITLE (Include Security Classification)

Parts-R-Us

12 PERSONAL AUTHOR(S)
Erik Brunvand

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Interim	FROM _____ TO _____	1987 December	43

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Parts-R-Us is a chip that contains a collection of building block parts for asynchronous circuit design. The parts contained on the chip are either not available as standard commercial components, or are standard gates combined into small modules that are particularly useful for building asynchronous control circuits. There are eight different configurations of Parts-R-Us, each offering a different set of asynchronous parts to the user. The parts contained on the chip include C-elements, transition call modules, transition selectors, transition toggles, transition arbiters, a four phase mutual exclusion element, an asynchronous register, two phase Q-registers, and four phase Q-registers.

This document is both a description of Parts-R-Us, and a user's manual for designers using the chip.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified

22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
Chahira M. Hopper	(513) 255-7865	AFWAL/AAAT-3

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

LIST OF FIGURES

TABLE OF CONTENTS

1. Introduction

As systems become larger, faster, and more complex, timing problems become more and more severe. Asynchronous systems may provide major advantages over traditional synchronous systems by avoiding the difficult problems of clock distribution, clock skew, and tuning that synchronous systems require. Asynchronous systems may be more versatile because they can adapt to and take advantage of improved subsystems that operate either faster or more slowly than the parts they replace and can take advantage of the maximum speed available from each of their parts. For example, many complicated integrated circuits cannot take advantage of incremental improvements in the manufacturing process because the timing relationships between parts of the circuit will be changed. Asynchronous techniques used for VLSI may eliminate this problem and let chips take full advantage of process improvements without changing the circuit. Asynchronous systems may be designed as a set of separate parts with simple communication protocols between them. This allows the designer to compose parts based only on their functionality, not on their timing properties. Asynchronous systems are a good match for parallel architectures since events are ordered by causal relationships among the elements of the system rather than synchronized to a global clock. As chip size and transistor count increase, wiring delays on the chip become significant, clock tuning becomes extremely difficult, and because previously designed modules are not as reusable even the cost of redesigning an existing chip increases. Thus as chip size and transistor count increase, the advantages of asynchrony also increase.

At present, however, asynchronous systems are rare despite the fact that they seem to offer significant advantages over traditional synchronous design in some cases. Asynchronous design is inconvenient and uneconomical since most commercially available parts are best suited for synchronous design. Because of the unavailability of suitable components, a lack of experience with such systems, and a lack of a simple, proven design methodology, asynchronous design is considered too difficult by most designers.

Parts-R-Us is a chip that contains a collection of building blocks for asynchronous circuit design. The parts contained on the chip are either not available as standard components at all, or are standard gates combined into small modules that are particularly useful for building this type of system. There are eight different configurations of Parts-R-Us, each offering a different set of asynchronous parts to the user. The parts contained on the chip include: C-elements, transition call modules, transition selectors, transition toggles, transition arbiters, a four phase mutual exclusion element, an asynchronous register, two phase Q-registers, and four phase Q-registers.

Parts like the ones used in Parts-R-Us are also described in [6,5,4]. With only a few exceptions these modules use two phase transition sensitive signaling. Parts-R-Us is intended for building small or medium sized asynchronous control circuits for testing, experimentation, or teaching. This document is both a description of Parts-R-Us and a user's manual since the pin numbers of the parts are shown in the diagram of each configuration.

2. Configurations

Each of the configurations of Parts-R-Us offers a different set of parts to the user. The configuration is chosen by setting the select pins according to Figure 1. Each configuration consists essentially of one type of part but since some parts are likely to be used in combination with others, extra

Select ABC	*Parts Description*
000	Four Phase Q-Flops
001	Two Phase Q-Flops
010	Selectors
011	FIFO Register
100	Arbiter and Call
101	C-Types and XOR
110	Call modules
111	Toggle and C-Types

Figure 1: Select Pin Settings for Parts-R-Us Configurations

Pin Number	*Pin Description*
1	Vdd
2–9	Shared Output Pins
10	Master Clear*
11–20	Shared Input Pins
21	GND
22–28	Shared Input Pins
29,30,31	Select A B C
32–40	Shared Output Pins

Figure 2: Pin Assignments of Parts-R-Us

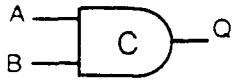
parts are included in some cases. For each configuration there is a full page figure in this document showing the parts available, the signal names, and the pin numbers.

Parts-R-Us is implemented in MOSIS 3μ CMOS and packaged in a 40 pin DIP. There are six pins common to all configurations, leaving 34 pins to be used by the various parts. These 34 pins are equally divided into input and output pins. The signal assignment of the 40 pins is described in Figure 2. The signal called Master Clear* in this figure is an active-low signal that clears the internal state of any parts that can be cleared. Outputs of clearable parts are 0 after a Clear.

2.1. C-Elements

Function

Muller C-elements, also known as "last-of" gates, or "rendezvous" elements, are well known to asynchronous circuit designers. A C-element's output follows the input only after all inputs have changed to the same value. That is, after all inputs go to 1, the output goes to 1. The output stays 1 until all the inputs have gone to 0, after which the output goes to 0 and so on.



C-Element

AB	Q_{n+1}
0 0	0
0 1	Q_n
1 0	Q_n
1 1	1

Enabled C-Element

$AB\overline{E}$	Q_{n+1}
X X 1	Q_n
0 0 0	0
0 1 0	Q_n
1 0 0	Q_n
1 1 0	1

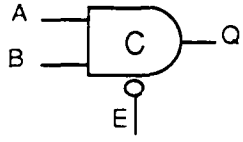


Figure 3: Truth Tables for C-Elements

There is also an enabled form of a C-element with an additional enable input. When the enable input is active the circuit behaves as a normal C-element, but if the enable input is not active the output of the C-element will not respond to changes on the input. Truth tables for regular and enabled C-elements are shown in Figure 3.

Design

A dynamic inverting C-element can be built from four CMOS transistors. The output node is pulled up if both signals are low, and down if both signals are high. With any other inputs the last output value is stored as charge on the capacitive output node. The C-elements on Parts-R-Us use a circuit like this and add an inverting negative resistance to hold the output at the last value is was driven to. The circuit is shown in Figure 4. Note that this circuit includes an enable input and an active-low master clear signal. The negative resistance is designed to provide weak drive that may easily be overdriven by the output of the dynamic C-element. This is done by using long narrow transistors to limit current to the output of the negative resistance. Rather than make the dynamic C-element drive the long gates of the negative resistance, the output is connected to two minimum size transistors in series with the long transistors. When cleared, the outputs of these C-elements will be forced to 0.

Application

A C-element is an example of a simple gate that is not available as a commercially available part, but that is used by almost all asynchronous circuits. With the exception of the asynchronous latch described in section 2.6., every part on Parts-R-Us includes a C-element somewhere in its implementation. Many of these are slightly modified versions of C-elements with or without enables, with inversions of some inputs, inverted output, or a different number of inputs.

Most of the variations on a basic enabled C-element can be built with the addition of a two input exclusive-or (xor) gate. This combination of gates, shown in Figure 5, will be called a C-type. Figures 6 and 7 show how a C-type may be used to implement every type of C-element used in the examples of [5].

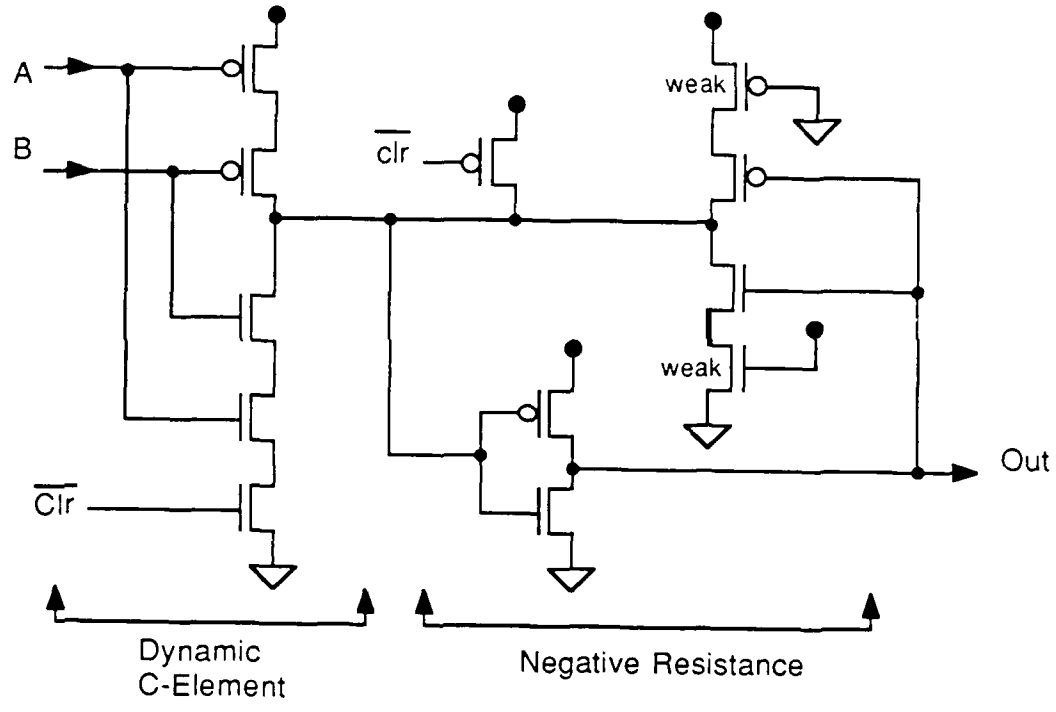


Figure 4: Schematic of an Enabled C-Element

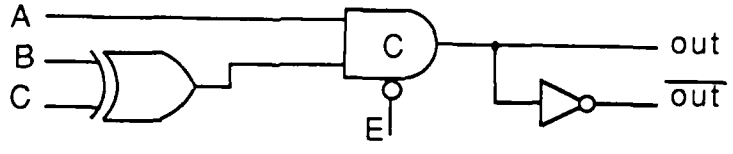
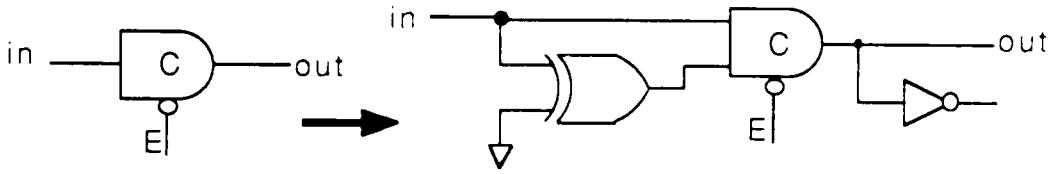
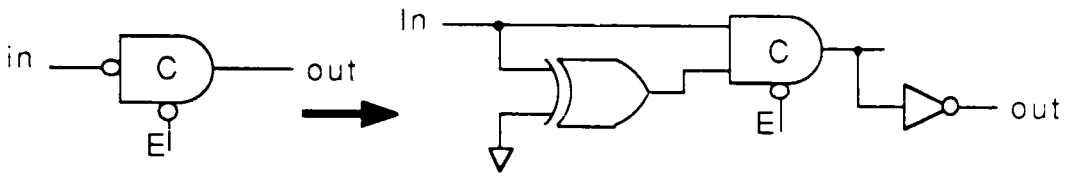


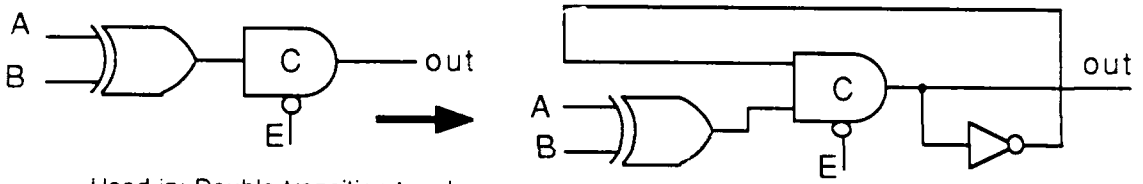
Figure 5: A C-Type General C-Element Module



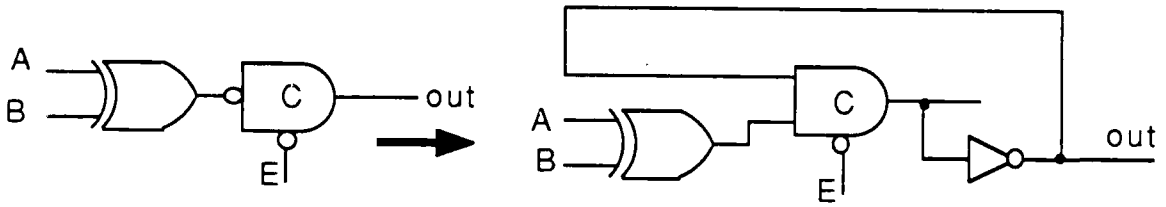
Used in: Transition toggle, 2PT to 2PB converter, 3-wire transition FIFO, Walton



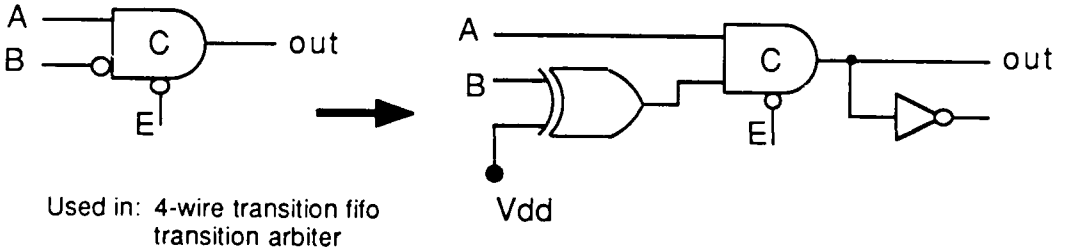
Used in: Transition toggle



Used in: Double transition toggle
Output selector



Used in: Double transition toggle



Used in: 4-wire transition fifo
transition arbiter

Figure 6: Different Enabled C-elements from C-types

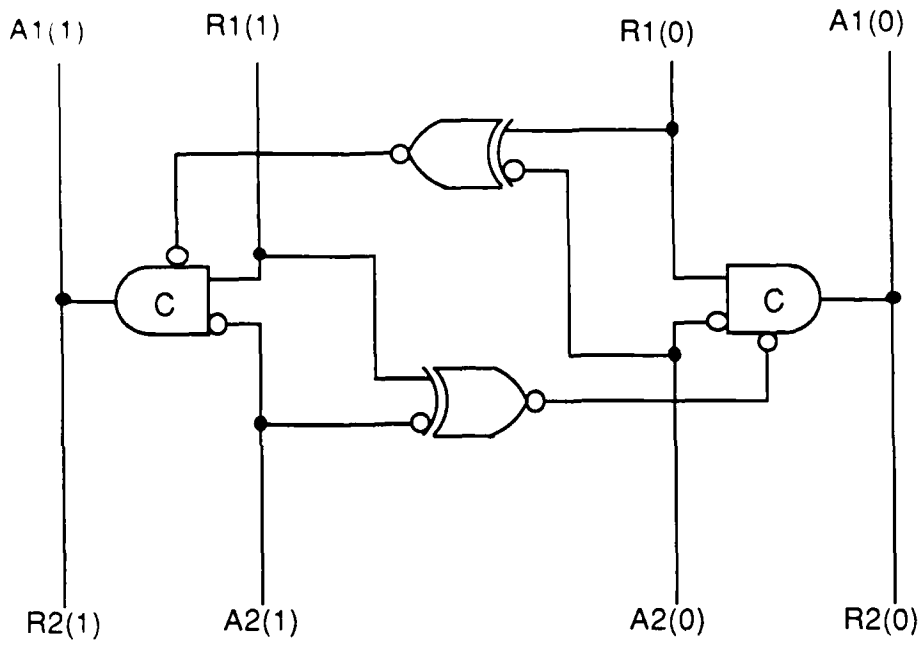


Figure 8: A Four-Wire Transition FIFO Cell

One simple circuit that can be built from two C-types and an xor gate is a call module. Call modules are described in section 2.2. and can be implemented as shown in Figure 10. This is an example of explicit use of the xor contained in the C-type.

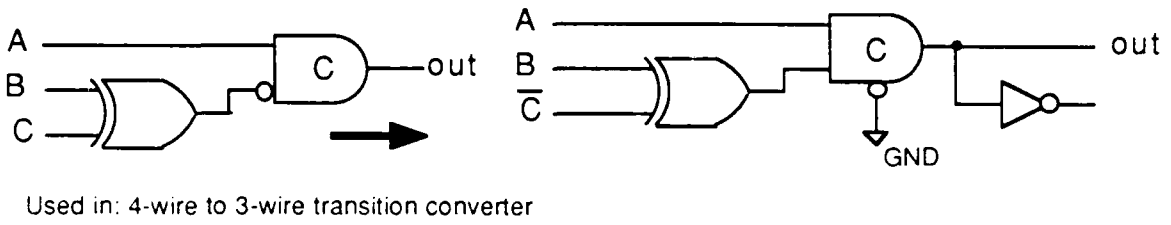
An example of a circuit that needs two extra xor gates, and uses the C-type to emulate a C-element with inversion on one input, is the four wire transition FIFO cell shown in Figure 8. A four wire transition FIFO cell has separate request and acknowledge wires for 0 and 1.

A "waiton" gate is an enabled C-element with only one input. If the enable is asserted then the output follows the single input, but if the enable is not asserted then the output remains in whatever state it was in. Another way to view this is that if the enable is *not* asserted then the waiton will wait until that signal *is* asserted at which time it will pass the information at the input to the output. Waitons are often used to hold up a transition until some signal is at a particular level. For example, a waiton with an active-low enable will hold up transitions on its input until the enable level is low.

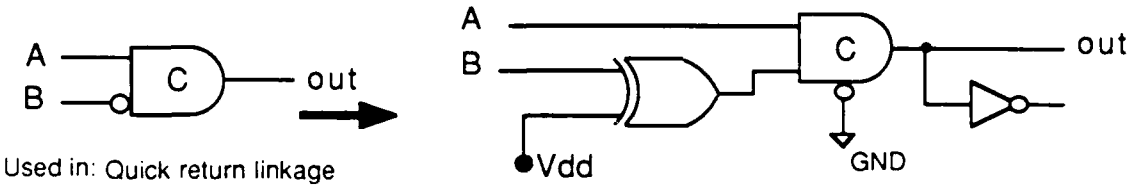
Configuration

The C-element configuration of Parts-R-Us includes four C-types and is shown in Figure 9. A C-type is a module that has twice as many inputs as outputs (4 inputs, 2 outputs) so half of the available output pins could go unused. Instead, the unused output pins will be used as alternative combinations of the inputs such that other interesting parts can easily be built from the C-types.

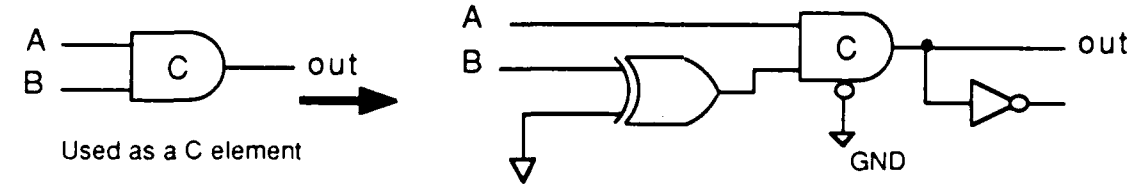
The first extra combination connects the outputs from pairs of C-types to the inputs of a simple two input C-element allowing the C-types to be used as four input C-elements. Going one step further, the outputs from the new four input C-elements are connected to another simple C-element



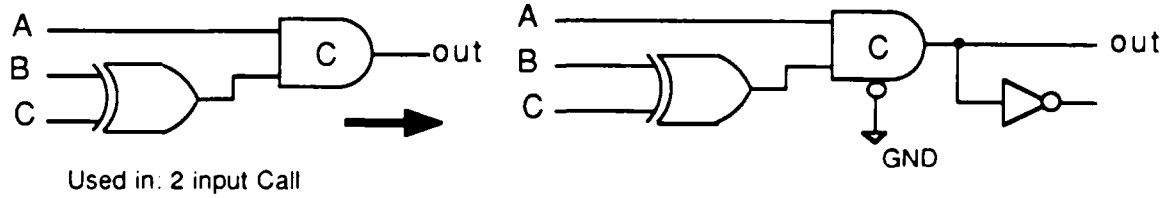
Used in: 4-wire to 3-wire transition converter



Used in: Quick return linkage
fifo control
etc....



Used as a C element



Used in: 2 input Call

Figure 7: Different C-elements from C-types

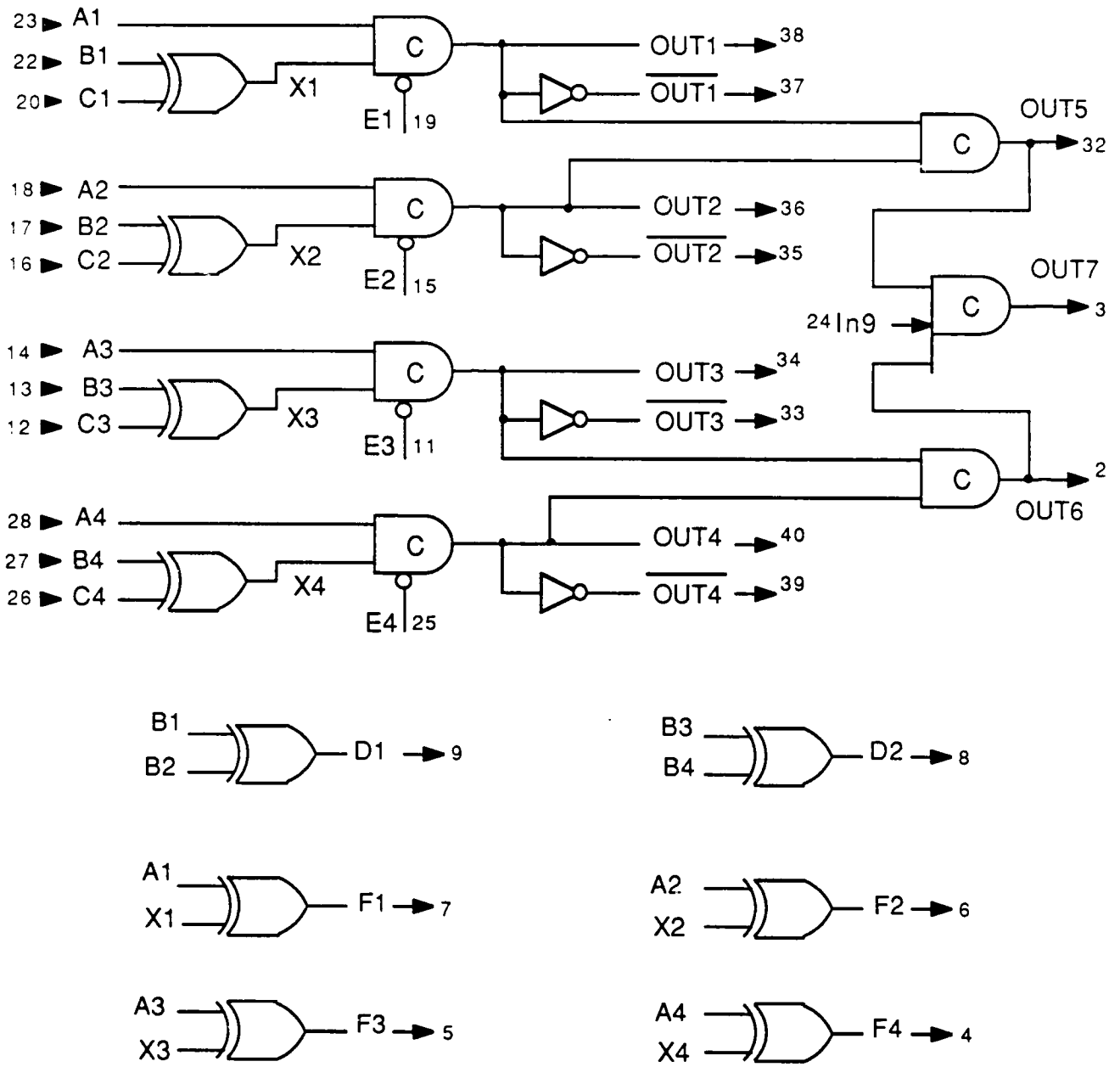


Figure 9: C-Type Configuration

to make an eight input C-element. Since there is still one extra input pin, the last C-element is actually a three input C-element allowing a nine input C-element to be used by combining all four C-types and the extra input. Since connecting two inputs of a C-element together has no effect on the function of the gate, C-elements with all number of inputs from one to nine can be used in this configuration.

Other extra logic is motivated by the example of a call module as seen in Figure 10. Since only an extra xor gate is needed to build a call module from two C-types, that extra xor is included in this configuration. The xor gates for building call modules have outputs labeled *D1* and *D2* in Figure 9.

The xor gates with outputs labeled *F1, F2, F3* and *F4* are designed to work with the C-types of the same number to implement the four wire transition FIFO shown in Figure 8. Any two C-types with their associated xor gates can implement a four wire transition FIFO cell. Actually, this doesn't work quite work as well as planned. Since the gates are exclusive-or, and the C-types are enabled low, extra inverters are needed to implement the circuit in Figure 8. The gates on Parts-R-Us really should have been exclusive-nor gates.

2.2. Transition Call Modules

Function

A transition call module can be thought of as the hardware equivalent of a subroutine call. In response to a request on either of the two request lines *R1* or *R2*, the call module starts a "subroutine" with the *Rs* signal. When the subprocess completes and acknowledges with *As*, the call module acknowledges the appropriate requester on *A1* or *A2*. It is the responsibility of the call module to remember which of *R1* or *R2* made the request so that the acknowledge gets routed to the correct requester. A full request/acknowledge transaction must complete before either side may request again. It is a mistake to assert more than one request concurrently.

It is also possible to view these call modules as decision-wait modules by ignoring the *Rs* signal. A decision-wait module will hold up a transition on one of its inputs until a transition arrives on *As*, then route the signal to the appropriate acknowledge.

Design

An implementation of a two input call module is shown in Figure 10. As noted in section 2.1. this circuit can be built from two C-types and an extra xor gate. However, because the C-elements used in a call are not enabled, regular non-enabled C-elements are used to implement call modules on Parts-R-Us. Since the C-elements clear their outputs to 0 after a Clear signal, call modules need no other clear input.

A call module remembers who made the request by using the storage inherent in a C-element. When a request is made on *R1* or *R2* one input of the appropriate *A1* or *A2* C-element is changed. When the subroutine acknowledge *As* occurs it changes the other input of the C-element to match, thus causing a transition on the correct acknowledge signal. The subroutine acknowledge also resets the other C-element to the state it was in before the transaction. The call module is now ready to

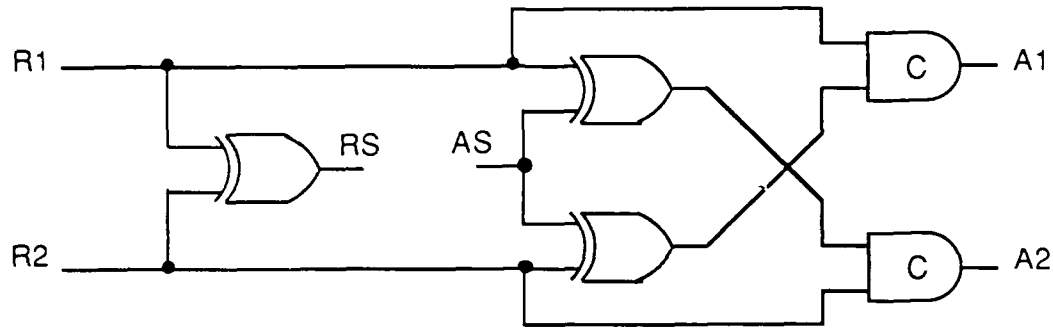


Figure 10: A Two-Input Transition Call

accept another request on either *R1* or *R2*.

Application

Call modules are usually thought of as providing a mechanism for implementing hardware subroutine calls. They can be used whenever a process or circuit is used more than once or by more than one requester. If, for example, the same input port must be queried for data in four different states of the control, a four-way call module with the input port as the subroutine process would let each state that needs input simply request to that call module to query for new data. The main restriction on the use of call modules is that requests to a call must not be concurrent.

Simple two-way call modules can be cascaded to make larger call modules by letting the subroutine process of a two-way call module be the inputs to another two-way call module. A tree of call modules can be built up in this manner to provide many callers with access to the same shared process.

When a call module is cleared, all outputs will be driven to 0 and the next action must occur at one of the two user request lines *R1* or *R2*. There is a variant of a call known as a calli that is used, in some sense, backwards. After a Clear the outputs of a calli will be pulled to 0 but the first action after a Clear is expected to be on the subroutine acknowledge *As*. This will cause a transition on the *A1* signal. From this point on the calli acts like a standard call module. The initial condition is the only real difference between call and calli modules. You can actually make a call into a calli by adding an inverter to the request input you want to be acknowledged first after a Clear, and an inverter to the subroutine request as shown in Figure 11. This fools the call into thinking that there is already a pending request that needs acknowledgment and so the first *As* event results in the proper acknowledge signal.

Configuration

This configuration, shown in Figure 12, consists only of call modules. It contains four two-way call modules and one four-way call. All of the call modules are normal calls. There are no calli modules since they can be easily constructed with the addition of two inverters.

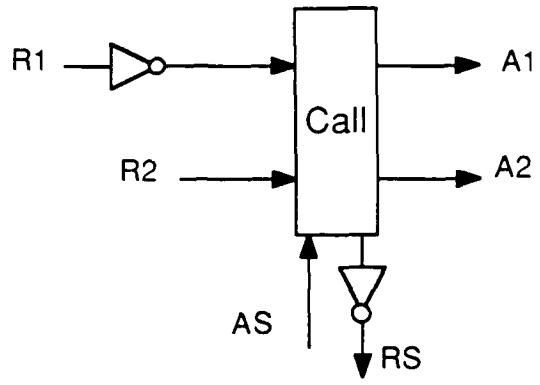


Figure 11: A Call Module Acting as a Calli

2.3. Transition Select Modules

Function

A two-way transition select module will steer a transition from its transition input to either output depending on the value of the select input. The select input is a level rather than a transition and must meet the constraints that the select input be valid before a transition occurs on the input and the transition must appear at the selected output before the select input can be changed.

Design

A circuit for a two-way Selector is shown in Figure 13. The select input enables one of the C-elements to steer the incoming transition to that output. The outputs are fed back to exclusive-or gates to make sure the output transitions happen correctly. That is, when an input transition occurs, the feedback from the selected C-element to the other exclusive-or removes the effect of the input transition from the unselected channel. Two transitions on the inputs of an exclusive-or return its output to the same state as before the transitions.

Application

Select modules can be thought of as a hardware implementation of an "if then else" statement in software. *If* the select input is high (true), *Then* steer the transition to the top output *Else* steer the transition to the bottom output.

Select modules are often used to enable some process only if some condition is true and simply to continue otherwise. Used in this way it is handy to have one of the outputs of the selector attached to an exclusive-or gate. This allows the acknowledge from the process enabled by the *Then* part of the construct to be combined with the *Else* part of the selector and thus acknowledge the entire construct as shown in Figure 14.

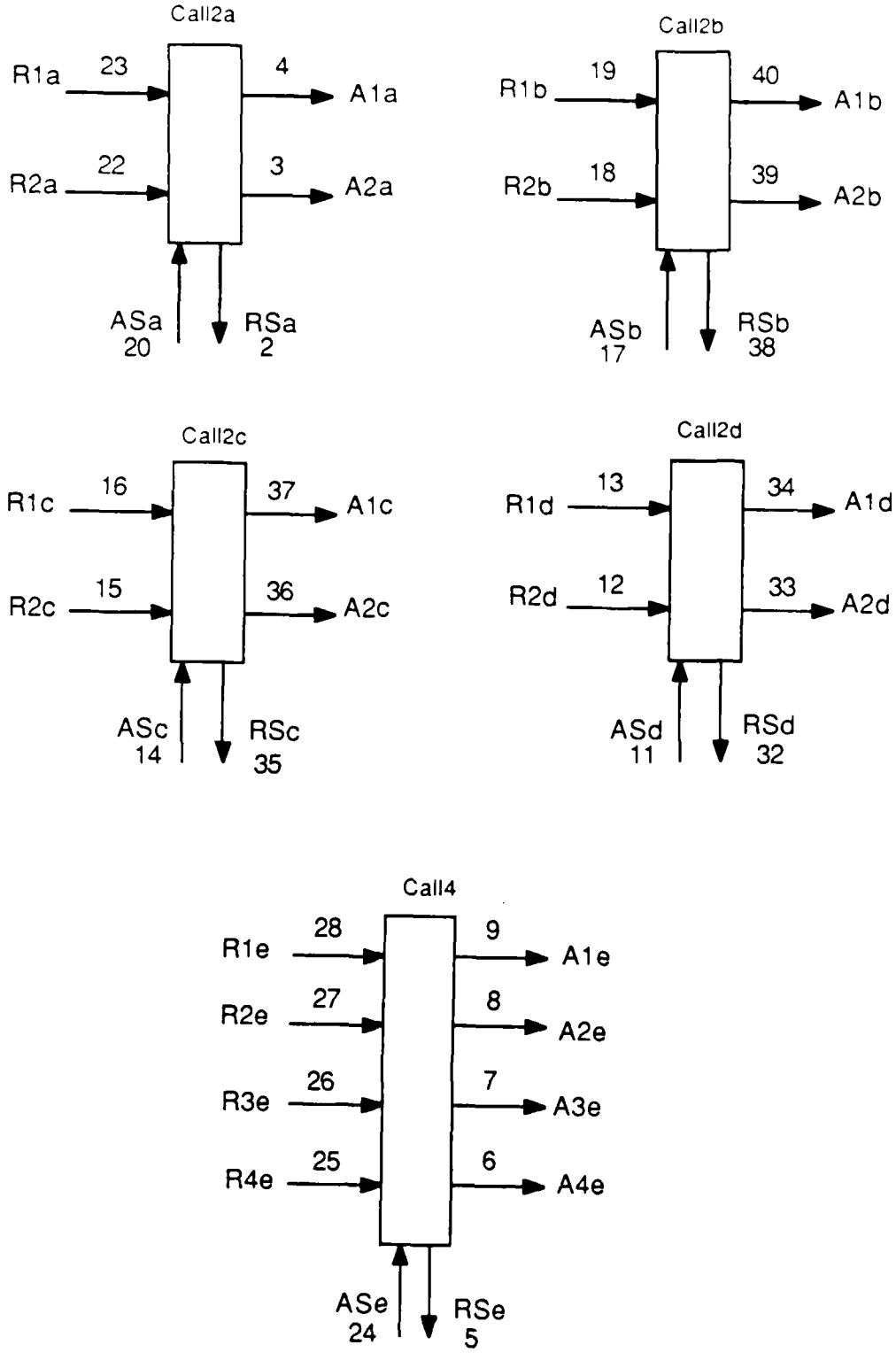


Figure 12: Call Configuration

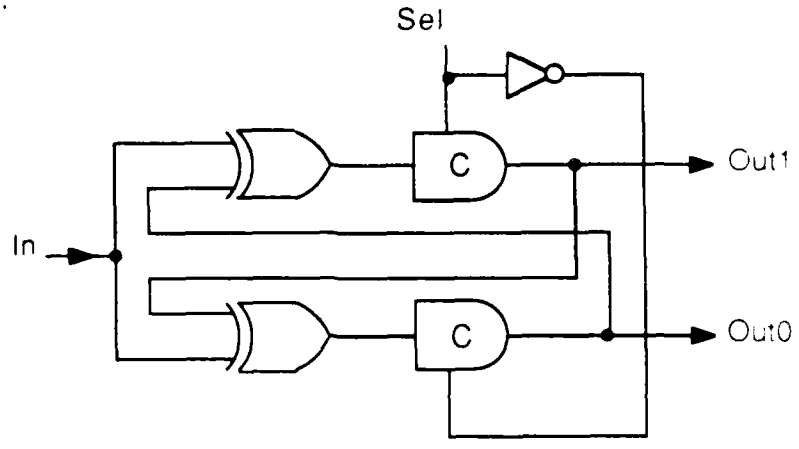


Figure 13: A Two-Way Transition Select Module

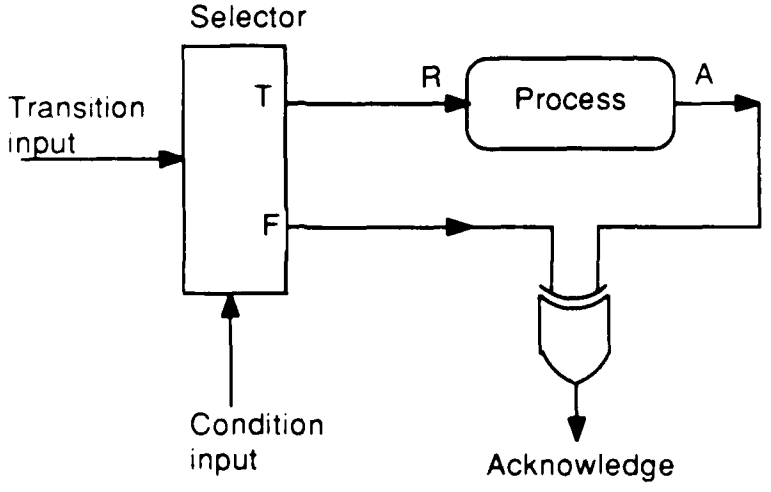


Figure 14: Selector and Exclusive-Or Combination

Configuration

The selector configuration, shown in Figure 15, consists of five two-way selectors and one four-way selector. As seen in the figure, four of the two-way selectors have exclusive-or gates on one output to help implement the structure described in the Application section. The four-way selector is implemented as a tree of three two-way selectors.

2.4. Transition Toggle Modules

Function

A transition Toggle module accepts transitions at its input, and sends the transition alternately to its two outputs. After a Clear signal the first input transition will be routed to the *Out0* output and subsequent input transitions will be routed to alternate outputs. When a toggle module is drawn as a circuit element the output that receives the first transition after a clear is noted with a dot as in Figures 18 and 19.

Design

A circuit for a two way Toggle is shown in Figure 16. An input transition will switch which C-element is enabled, thus letting a transition out one of the outputs, and in the process set up the next transition on the input of the C-element that is not enabled.

The circuit in Figure 16 may have a problem if the two C-elements are ever enabled at the same time. This may happen if the delay through the inverter on the input is significant. In this case the toggle may oscillate, which in a system that considers each transition to have meaning would be disastrous. The actual circuit implemented on Parts-R-Us is designed to avoid any possibility of oscillation by using a two phase non-overlapping signal generator on the input. This circuit, shown in Figure 17, uses C-elements with active-low enables. The extra circuitry makes sure that the signals are non-overlapping low so that it is never the case that the two C-elements can be enabled at the same time.

Application

Toggle modules are used anytime a circuit needs to cycle between two alternative courses. One simple example of this is using a toggle module and an exclusive-or gate to convert between two and four phase signalling. Converters to convert in both directions are shown in Figure 18. It is also fairly common to see call modules used such that the two users of the call alternate requests for use of the shared subroutine. In this case a toggle may be substituted for the call module.

Configuration

The Toggle configuration, Figure 19, includes five two-way toggles. Since a toggle has twice as many outputs as inputs, this leaves a lot of input pins to fill. Since C-types have twice as many

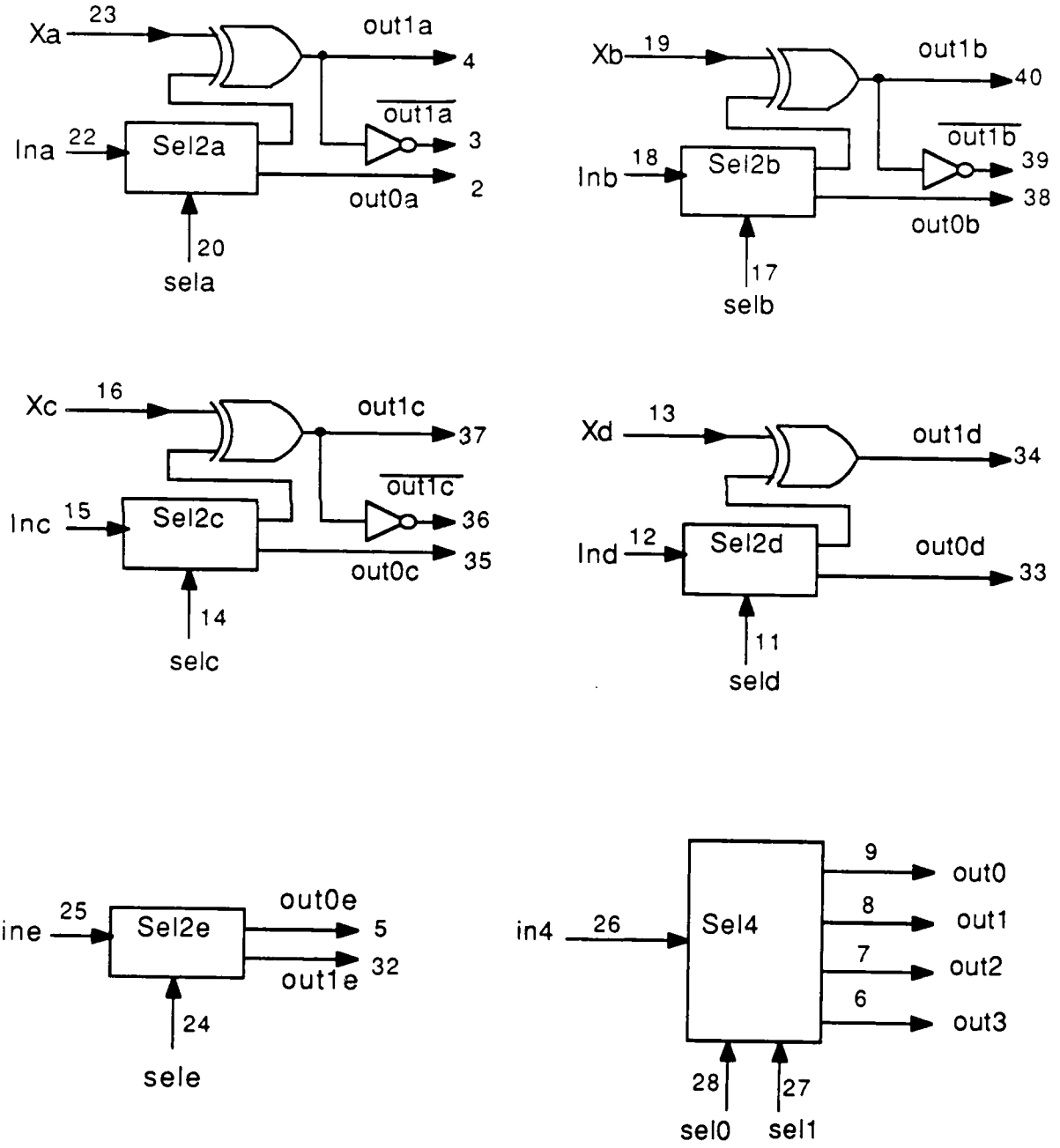


Figure 15: Selector Configuration

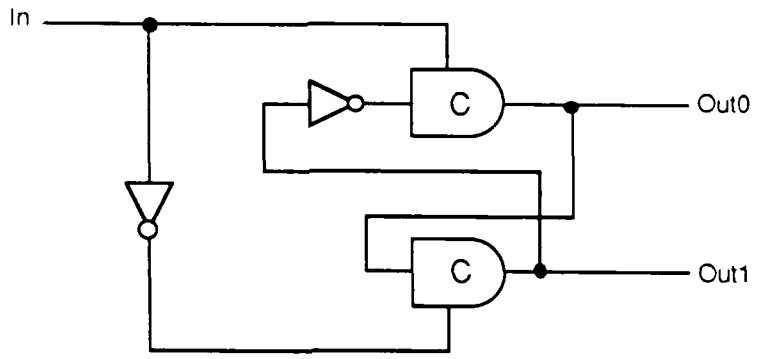


Figure 16: A Two-Way Transition Toggle

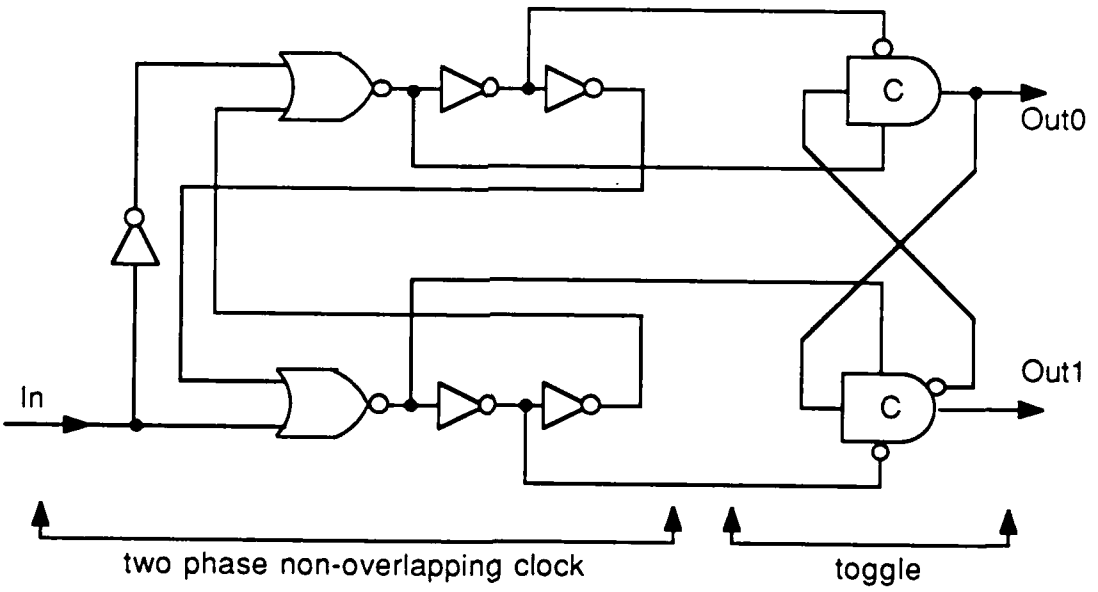


Figure 17: A Conservative Two-Way Transition Toggle

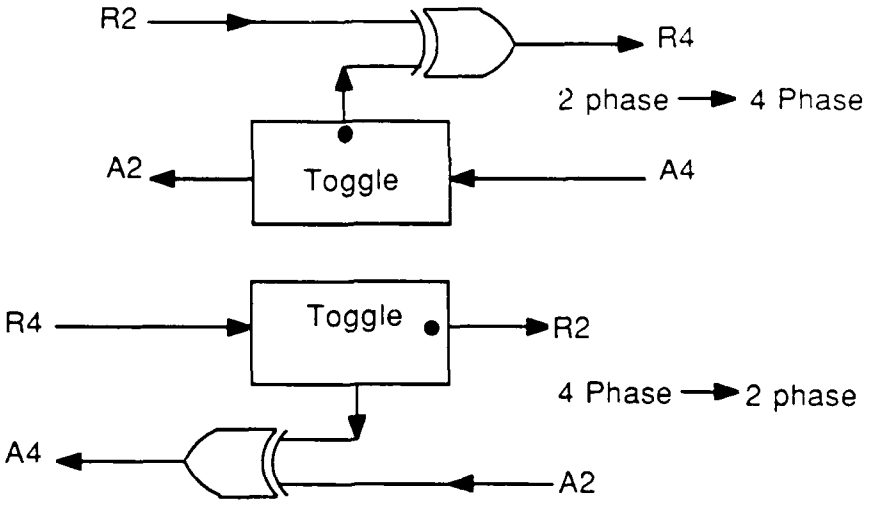


Figure 18: Converters for Two and Four Phase Signalling

inputs as outputs, they are used to fill up the spaces. Three C-types are included, as well as one extra C-element combining the outputs of two of the C-types. These are actually the same physical circuits as the first three C-types from the C-element configuration.

2.5. Transition Arbiter Modules

Function

An arbiter is a module that is used to guarantee mutually exclusive access to some shared or protected process. If a single user requests access by causing a transition on the *R1* or *R2* lines, access will be granted promptly and the shared process will be requested by a transition on the appropriate *G1* or *G2* wire. The shared process must declare itself done when it is finished by causing a transition on the *D1* or *D2* line and the arbiter then acknowledges the initial requester by making a transition on the *A1* or *A2* line.

Another way to view this transaction is that a user requests access to a shared resource by causing a transition on the *R1* line. The arbiter grants access with the *G1* signal. The original requester signals that it is finished with the shared resource by making a transition on the *D1* line, and the arbiter acknowledges that done signal with the *A1* line.

If another user requests access while the first has been granted access but has not yet finished, the second requester must wait. If more than one user requests access concurrently, one and only one requester will be granted access to the shared process, and the others must be delayed.

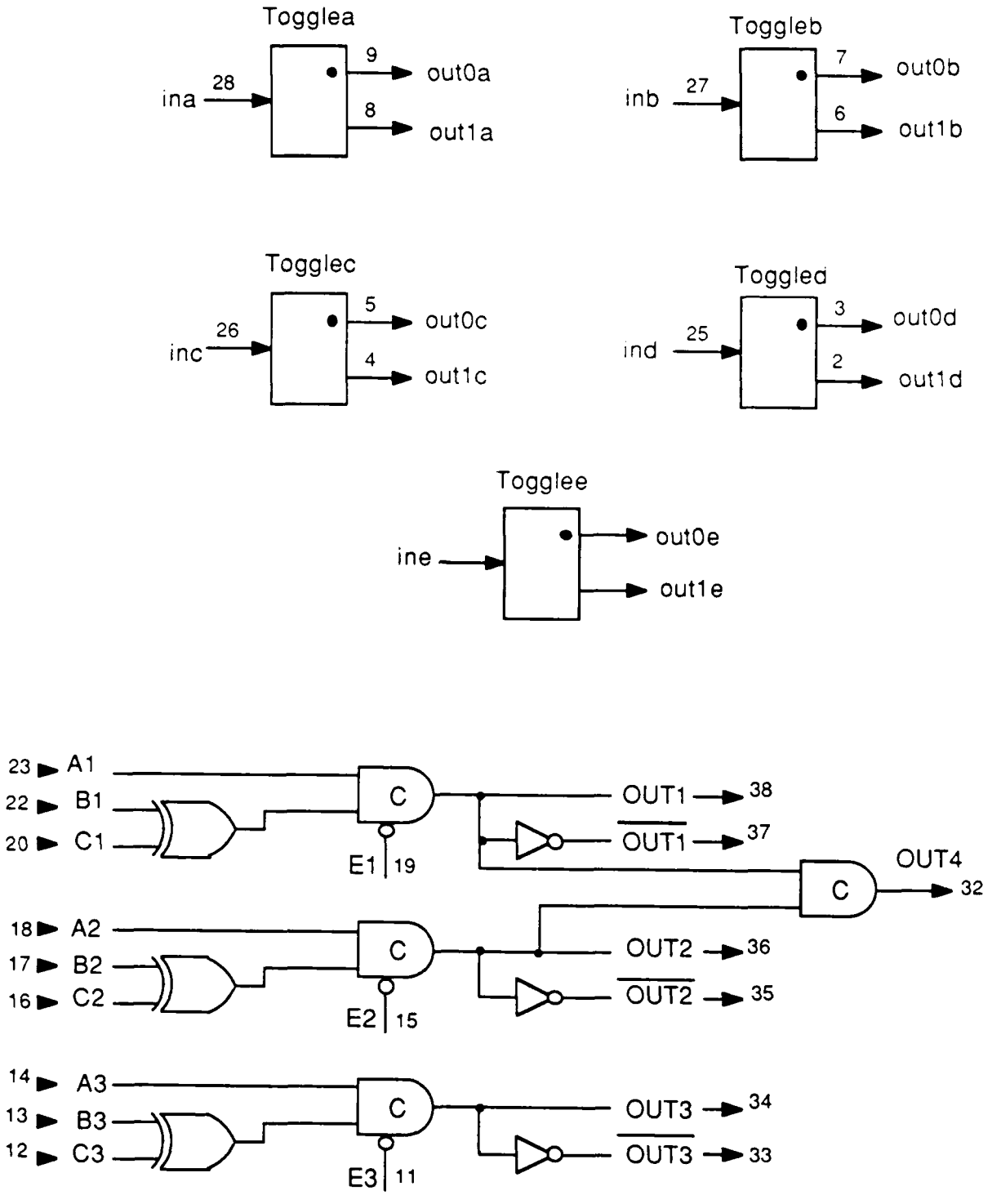


Figure 19: Toggle Configuration

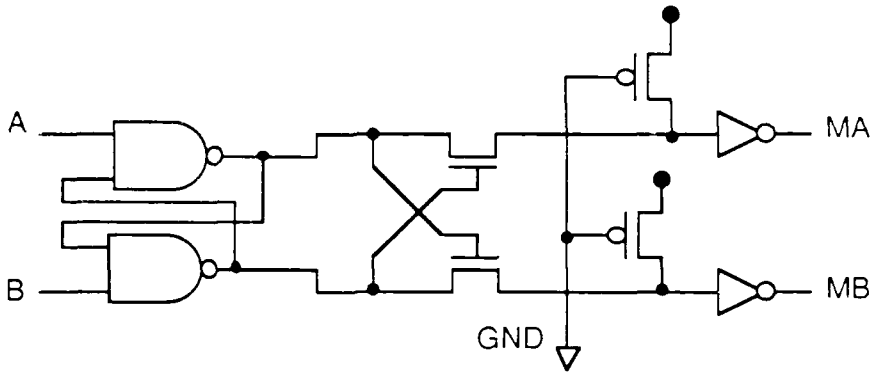


Figure 20: A Four Phase Mutual Exclusion ("mutex") Element

Design

In order to make a clean arbitration choice, a four phase mutual exclusion element, or mutex element, like the one shown in Figure 20 is used. This circuit is essentially a set/reset flip flop followed by a comparator to tell when the flip flop has been set. The problem is that both A and B may be asserted at the same time and the flip flop may enter a metastable state. When the flip flop has decided which way to flip, a flip flop output will go low. The comparator will not let either pass transistor turn on and pass a signal until the difference between the flip flop outputs is greater than a transistor threshold, which is a good indication that the flip flop has exited metastability. When this happens one of the nodes connected to the weak p-type pull-up transistors will be pulled to ground which asserts the appropriate MA or MB signal.

A two-way transition arbiter can be built easily using this mutual exclusion element. One circuit is shown in Figure 21. The exclusive-or gate recognizes when there is a request by sensing a difference between the Rn request and An acknowledge lines. This sends a signal into the mutual exclusion element. The mutual exclusion element chooses one of the inputs A and B and asserts the appropriate MA or MB signal. This enables the one-input C-element, or waiton, to let the Gn grant signal through. When the Dn "done" transition happens, the input to the mutual exclusion element is lowered and the An acknowledge transition gets through the other waiton.

Application

Arbiters are needed anytime concurrent requests to a single resource must be made. One particular instance where arbiters are often required is with call modules. Call modules, described in section 2.2., allow multiple non-concurrent requests to common processes. Used with an arbiter as in Figure 22 the requests may now be concurrent.

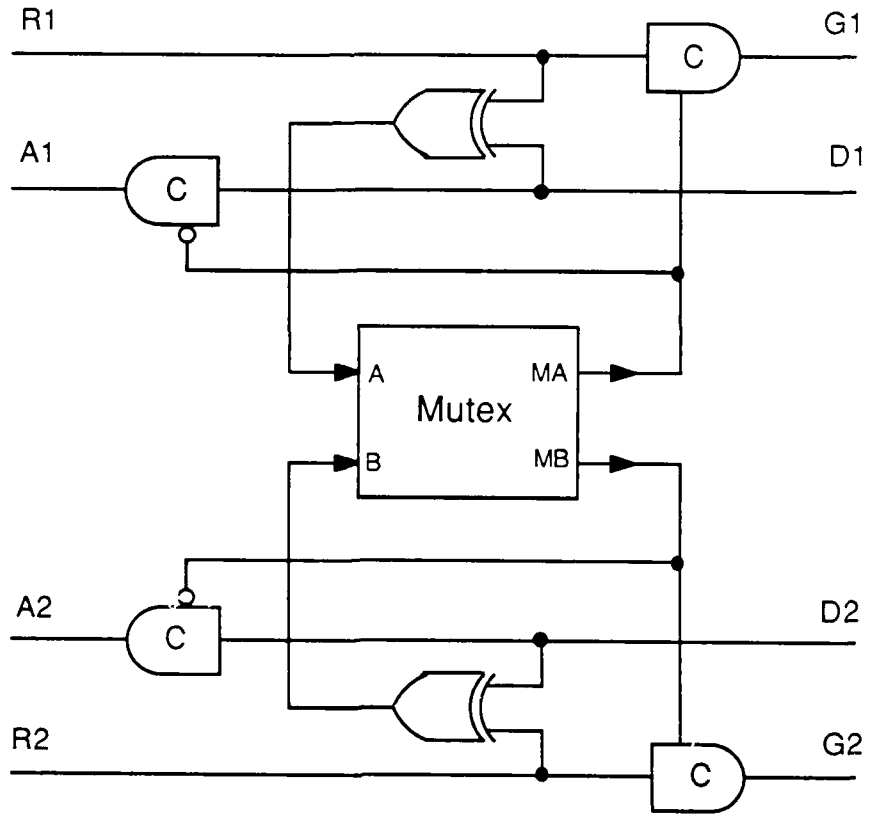


Figure 21: A Two-Way Transition Arbiter

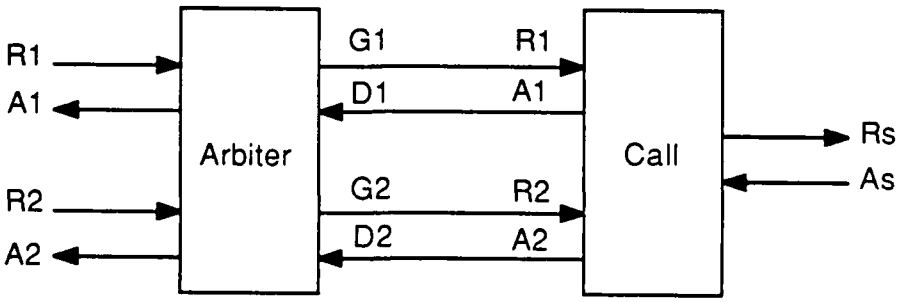


Figure 22: An Arbiter and a Call

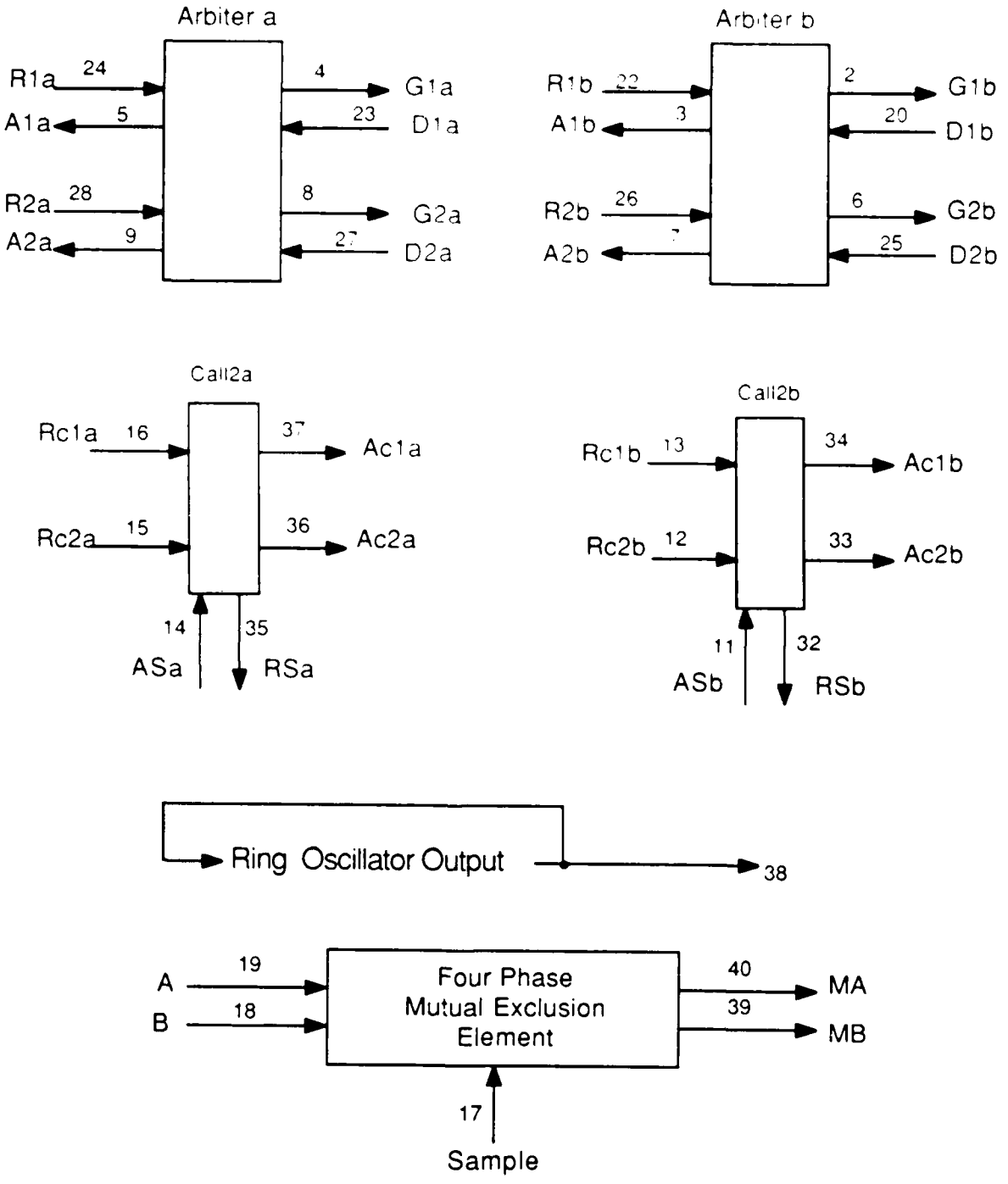


Figure 23: Arbiter Configuration

Configuration

This configuration, shown in Figure 23, includes two two-way transition arbiters and two two-input call modules since they are often used in combination.

Other bits and pieces included in this configuration are a four phase mutual exclusion element and a ring oscillator for timing and testing. The mutual exclusion element has a simple sample and hold feature so that the outputs may be sampled at different intervals from concurrent requests to help determine the metastable exit properties of the circuit. The ring oscillator is a string of 30 two-input C-elements with their gates tied together and an inverter at the end of the string. The ring will circulate only while the Clear signal is asserted so that no extra activity is going on during normal operation of Parts-R-Us.

2.6. FIFO Register

Function

An asynchronous latch has the usual request and acknowledge signals. A request is a signal to store the current input of the latch, and when that value is stored a transition is caused on the acknowledge line. Notice that although the control signals are transitions, the values being stored in such a latch are level values and must be stable at the time the latch request occurs.

An asynchronous FIFO (first in first out queue) is a sort of a latch that has two request-acknowledge pairs. One request is from some process to store a new value into the latch, and its acknowledgment means that the new value has been stored. The other is a request to some other process that there is new information in the latch, and the acknowledgment means that the other process is finished with that data. The sequence of operation for a FIFO begins when process1 requests the FIFO to latch new data. The FIFO latches new data, acknowledges that the data is latched, and causes a transition on the other request line offering this new data to process2. Only after process2 acknowledges that it is through with the data will the FIFO honor a request to store new data into the latch.

Design

The basic cell used to make both latches and FIFOs is shown as the part of the circuit inside the dotted box in Figure 24. The circuit is essentially two flip flops that each select whether to follow the input or latch the current input value. The pass gate selected by the X signal on the output of each flip flop determines which one will drive the output of the cell and the enabled inverters gated with the Y signal determine if the flip flop is following the input or holding a value. The two flip flops are wired such that if one is following the input, then the other must be latched, and only one of the flip flops may be driving the output at any time.

Application

The basic cell described in the previous paragraph can be used to make an asynchronous latch by connecting the *Xout* signal to the *Yin* signal as seen in Figure 24. A transition on the *Req* line will first switch the output to the flip flop that is following the input and then cause that flip flop to latch the current value. After this happens the transition is sent back to the requester as the *Ack* signal. The flip flop that was holding the previous value is now following the input ready to latch that input value when another *Req* transition occurs.

One way to build asynchronous FIFO from the basic cell is shown in Figure 25. In the quiescent state there is a connection from *D* to *Q* through either the top or bottom flip flop in the basic cell. A request from the previous FIFO cell on the left, *R1*, will latch the current input data into one half of the circuit, start sampling subsequent data on the other half of the circuit, acknowledge that the data has been latched on *A1*, and request the next cell on the right to pick up the latched data with *R2*. The acknowledgment *A2* from the next cell on the right signals that the data has been consumed, and switches the output to the other half of the circuit so that the new data appears at the output.

Notice that in order to build a FIFO from the basic cell an extra C-element is required. This C-element enforces the condition that the process on the right must acknowledge and the process on the left must have requested before new data can be latched. The inversion on one input of the C-element, shown as a bubble on one input in Figure 25, initializes the FIFO control so that the very first time a request comes from the left, new data can be latched.

Configuration

This configuration includes a nine bit wide set of basic cells with common control signals. The circuit enclosed in dashed lines in Figures 24 and 25 is the cell that is repeated, and the *Xin, Xout, Yin,* and *Yout* signals are shared for all nine cells. By connecting *Xout* to *Yin* this configuration can be used as a nine bit wide asynchronous register.

With the addition of an extra C-element as shown in Figure 25 this can implement a nine bit wide, one bit deep FIFO. So that the register may be used this way, two C-types are also included in this configuration as shown in Figure 26.

2.7. Q-Flops

Function

A Q-flop is a module that allows an asynchronous signal to be sampled on request, and generates an acknowledge only when a reliable data value is available at the output. Since the sampling circuit may enter a metastable state, the acknowledge must be delayed until the circuit has exited metastability and a reliable result is reported. Unlike the asynchronous latch of section 2.6., the data, being asynchronous with respect to the request, may be changing when the request to sample the value occurs. Q-flops were first described in [2] to use use four phase signalling and assign a meaning to every edge of the four-cycle exchange. A timing diagram is shown in Figure 27.

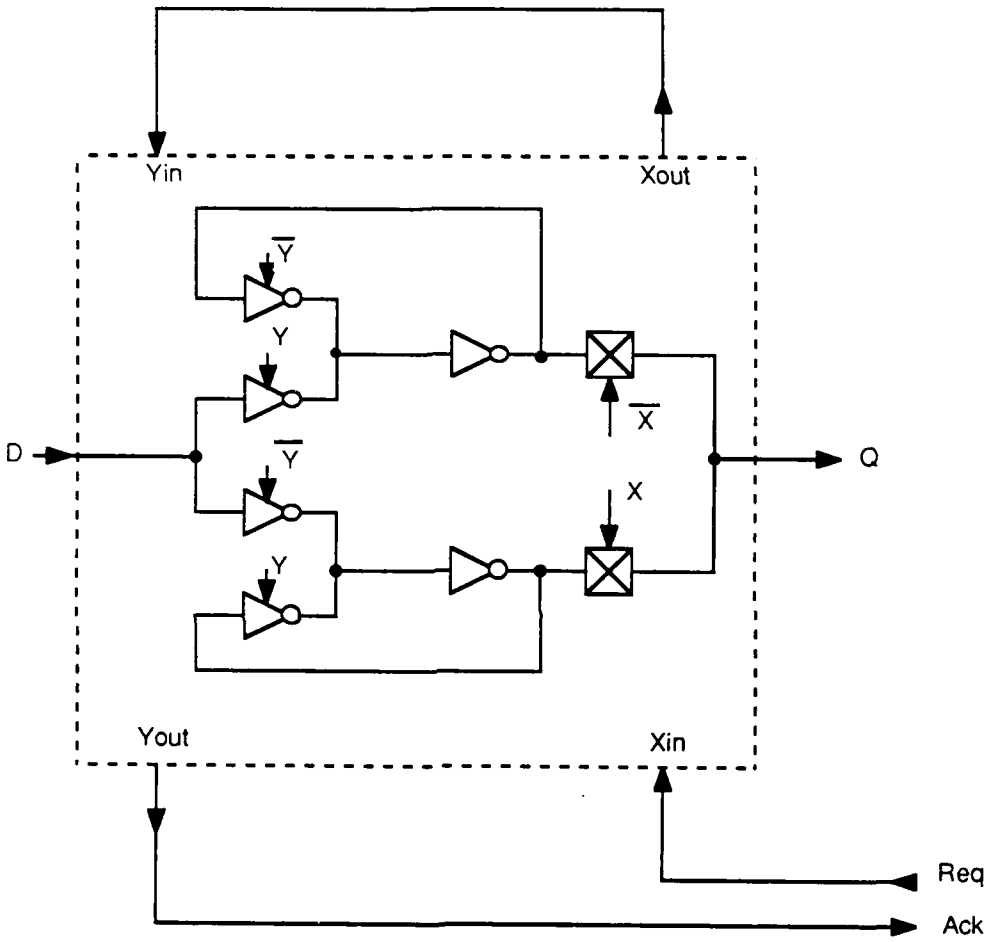


Figure 24: One Bit of Asynchronous Latch

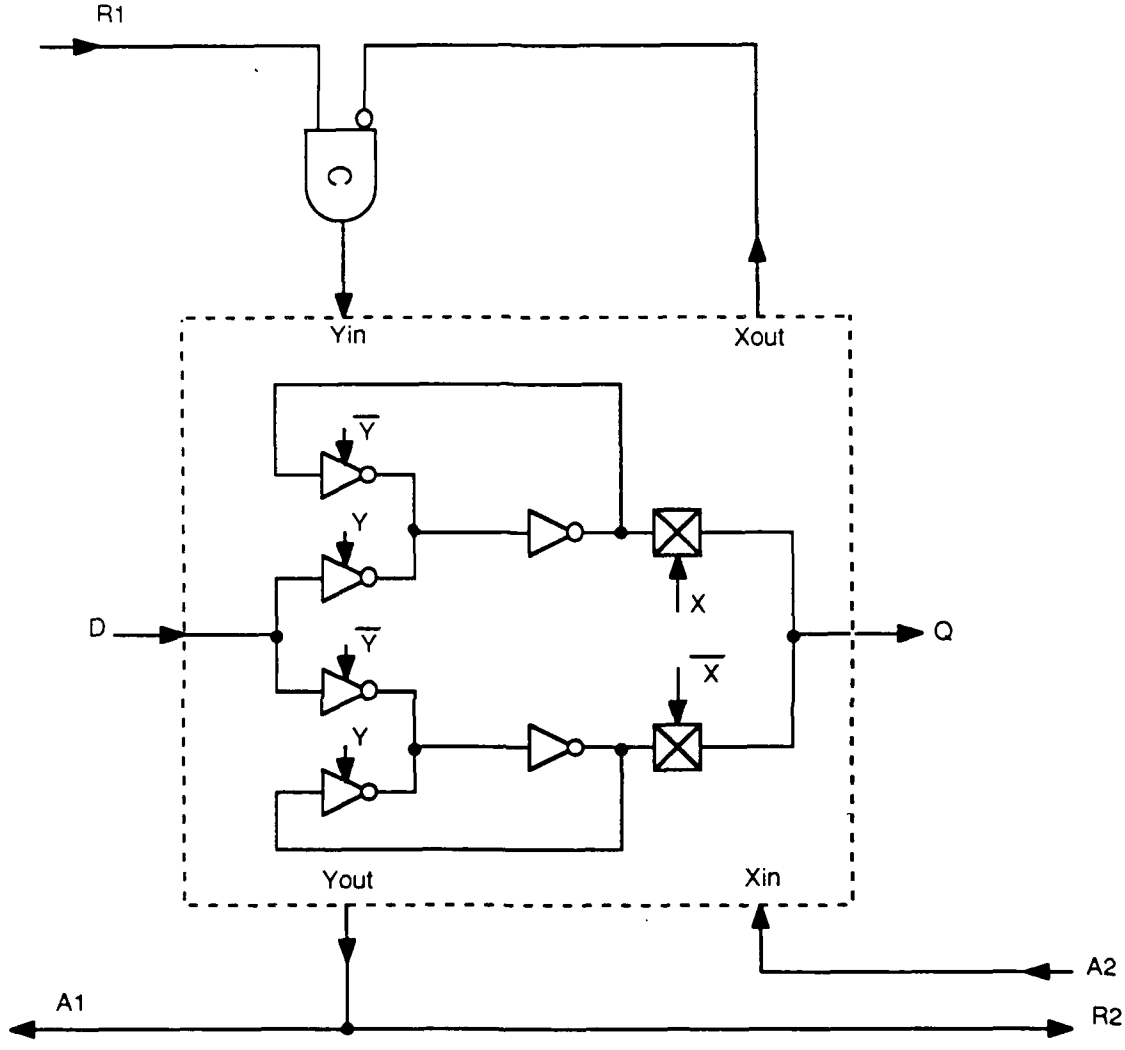


Figure 25: One Bit of Asynchronous FIFO

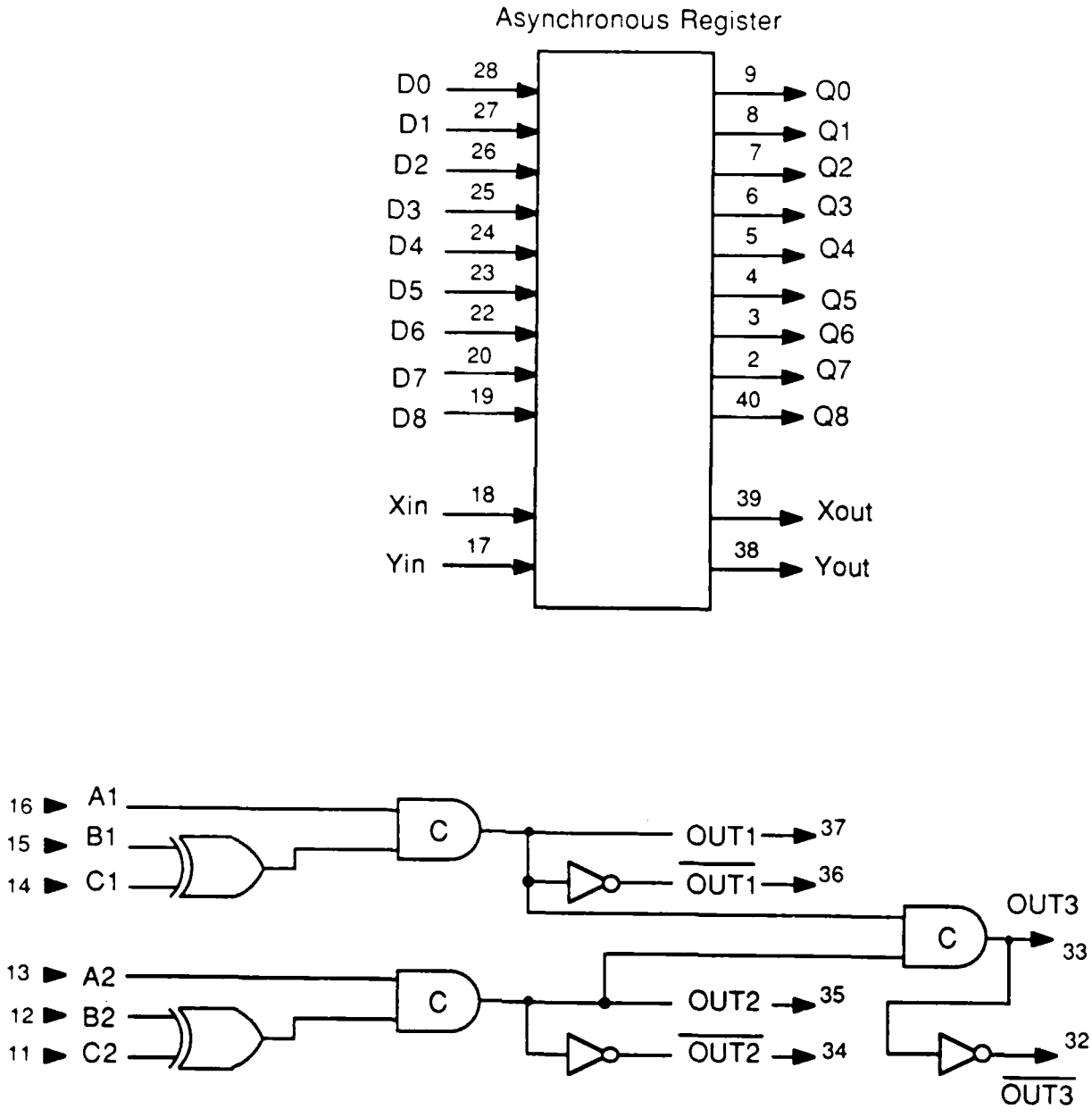


Figure 26: FIFO Register Configuration

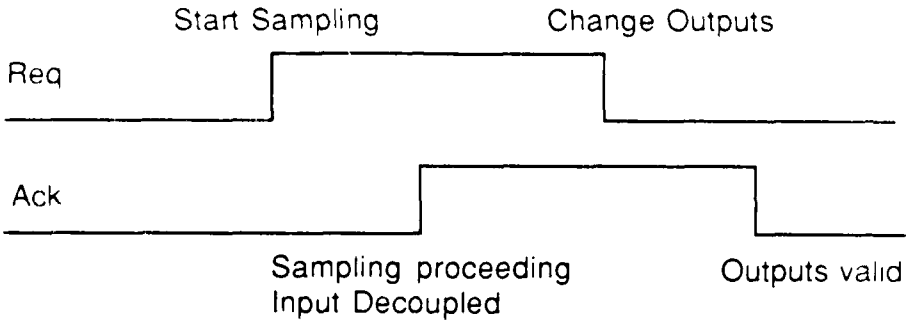


Figure 27: Original Q-Flop Timing

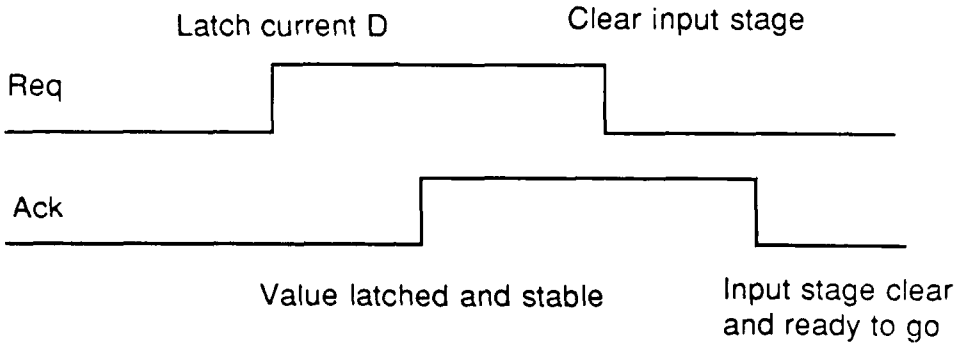


Figure 28: Parts-R-Us Q-Flop Timing

Design

The Q-flops implemented on Parts-R-Us, while performing the same basic function of sampling an asynchronous signal, behave a bit differently than the Q-flops described in [2]. The basic Q-flop on Parts-R-Us is still a four phase device and is broken into two basic units: the sampling stage, and the output stage. The timing diagram of this type of Q-flop is in Figure 28.

The circuit for the sampling stage is shown in Figure 29 and is further broken down into two parts: the input section and the comparator. The input stage is controlled by two signals *C1* and *C2* which are generated from a request. If there is no active request then both *C1* and *C2* are low. This will force the inverter between *A* and *B* to "eat its own tail." This will settle the voltage at both *A* and *B* to approximately half of the voltage swing of the inverter. This value also appears at the input of the enabled inverter enabled on *C2*. The input section will eventually form a flip flop from these two inverters which is now poised right at the metastable state because of the intermediate voltage at the inputs to the inverters. When a request to sample occurs the *C1* signal goes high, which routes the current *In* signal through the *AB* inverter to the input of the *C2* inverter. Then *C2* goes high to make the two inverters into a flip flop which latches the current *In* value, and also decouples the input from the flip flop. Actually *C1* and *C2* can be the same signal as long as *C1* changes before, or at least not after, *C2*. In this case the value that is being sampled is actually the value stored on the dynamic node on the output of the enabled inverter at

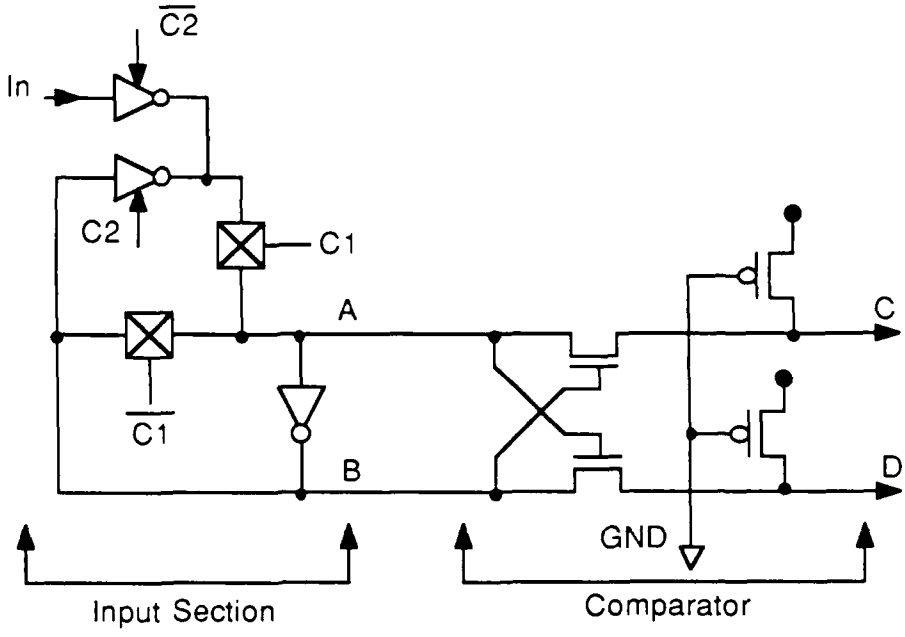


Figure 29: Q-Flop Sampling Section

the input.

The comparator connected to the input section flip flop is the same as the one used in the mutual exclusion element shown in Figure 20. The C and D signals will be held high by the weak p-type pull ups until the A and B signals differ by more than a transistor threshold value. Again, this is a good indication that the flip flop has decided on an output value and the corresponding comparator output should be pulled low. When the Q-flop is not sampling, the voltage at A and B will be the same.

Using this Q-flop sampling section it is easy to build a four phase Q-flop by adding an output stage, and some logic to generate the acknowledge. Shown in Figure 30, the output stage for a four phase Q-flop is a simple set/reset flip flop with a Clear input. When the input stage has sampled the input it will pull one of $\overline{QS}$ or $\overline{QR}$ low, causing the output stage to set or reset. When one of the set or reset signals has gone low, and the output flip flop has agreed on the value, then the Q-flop acknowledge is asserted. The timing diagram in Figure 28 shows the sequence of control in this Q-flop.

Four phase Q-flops are controlled by level signals while just about every other part on Parts-R-Us senses transitions. Luckily, there is an easy way to combine two four phase Q-flops into a single two phase Q-flop that, while it still samples a level at the input, now operates on transitions as control signals. A timing diagram for this circuit is shown in Figure 32 and the circuit is shown in Figure 31. It operates by alternately sampling into the top and bottom four phase Q-flops. When the request to sample arrives, one of the four phase Q-flops starts sampling, and the other, having been used the last time, resets its input stage. When the half that is sampling has a valid output and the half that is clearing is through clearing, the C-element switches the output to the sampled side and announces an acknowledge. Thus the two phase Q-flop will sample a new input on every

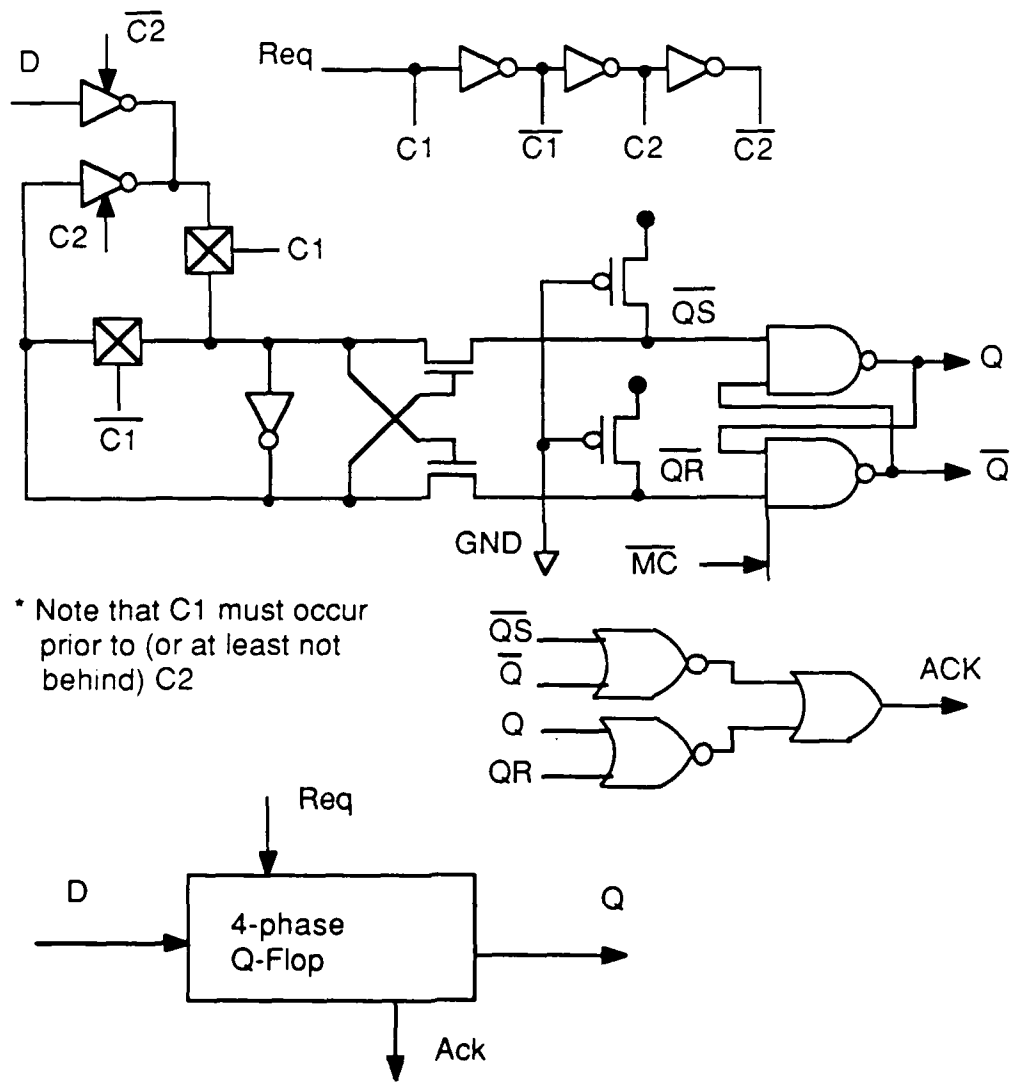


Figure 30: A Four Phase Q-Flop

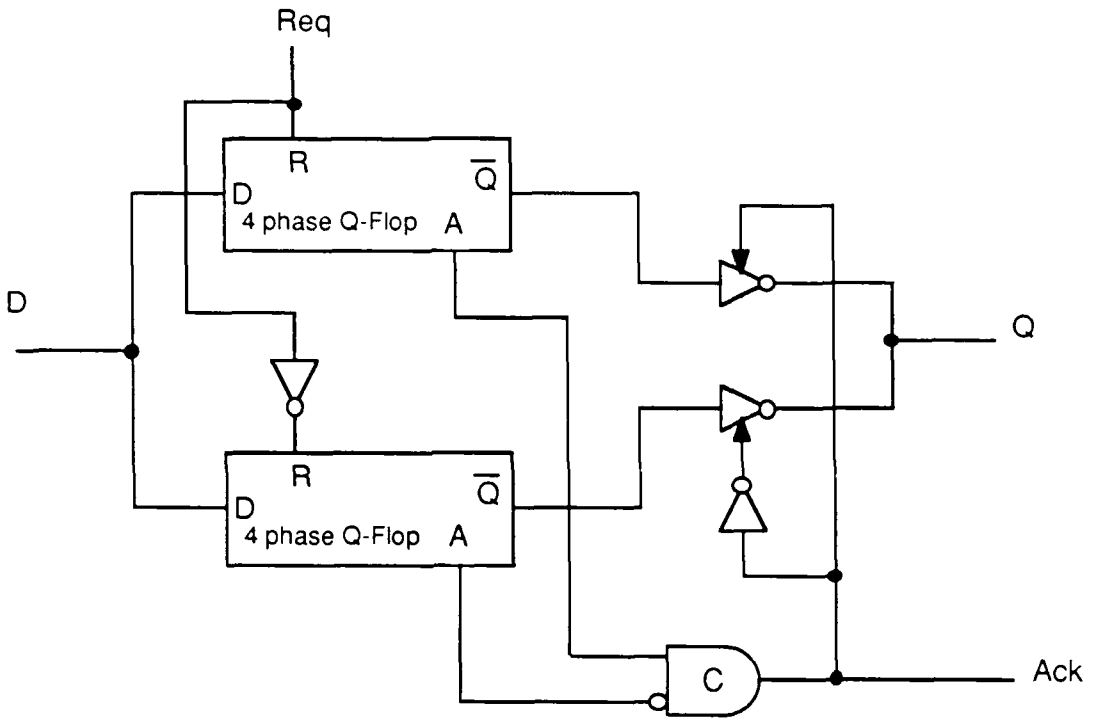


Figure 31: A Two Phase Q-Flop

transition of the control input.

Application

Q-flops can be used anyplace where an asynchronous signal needs to be sampled. In particular it is easy to build state machines with Q-flops that can include asynchronous inputs. A diagram of such a state machine is shown in Figure 33. This state machine will enter its next state only after the Q-register has acknowledged that all the outputs are valid, and after a delay that models the delay through the next-state logic. The Q-register in this type of state machine may be either two or four phase.

One way that the two phase Q-flop can be used is as a FIFO cell in much the same way as the FIFO cell in section 2.6.. Notice in Figure 34 how an extra C-element is needed, and that the signals exactly match those in Figure 25. The difference is that while the FIFO in Figure 25 required that the data be stable before requesting, the Q-flop version may be used in places where the input data is asynchronous with respect to the input request and sampling is desired.

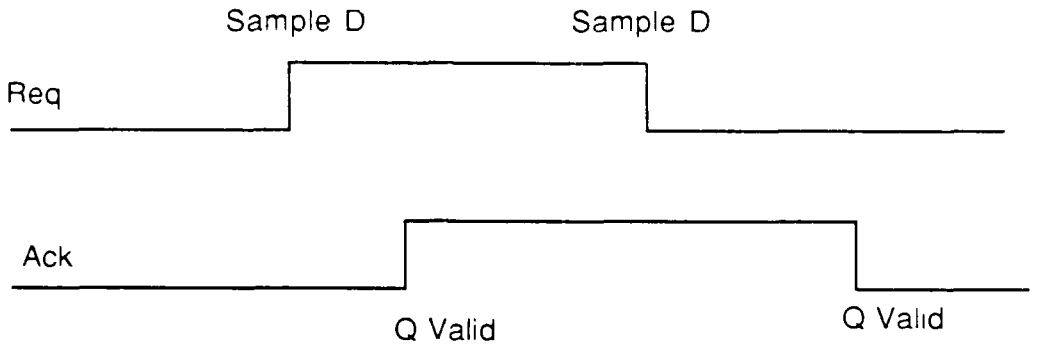


Figure 32: Two Phase Q-Flop Timing

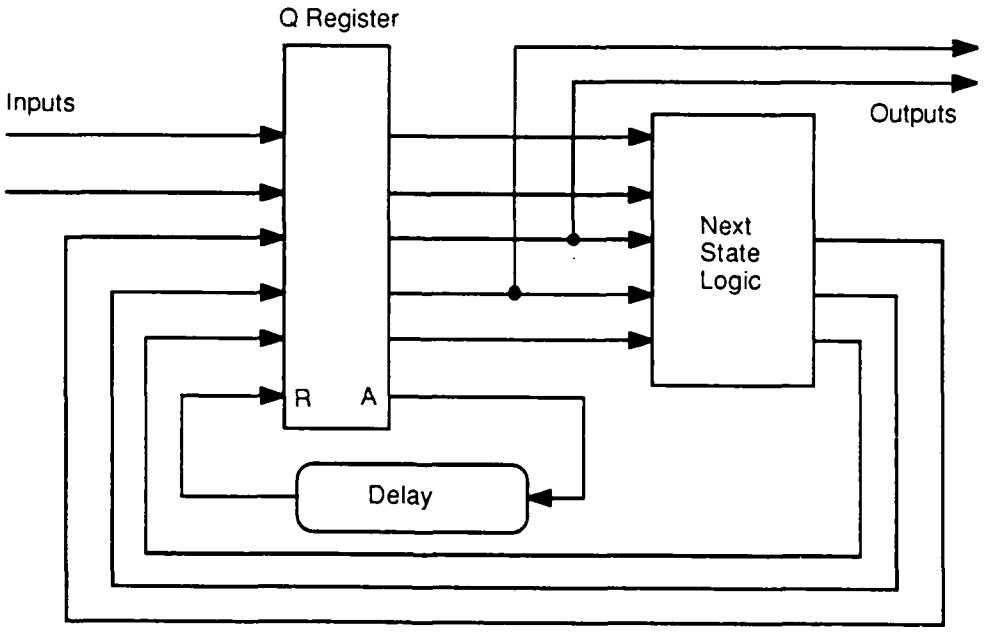


Figure 33: A State Machine Using a Q-Register

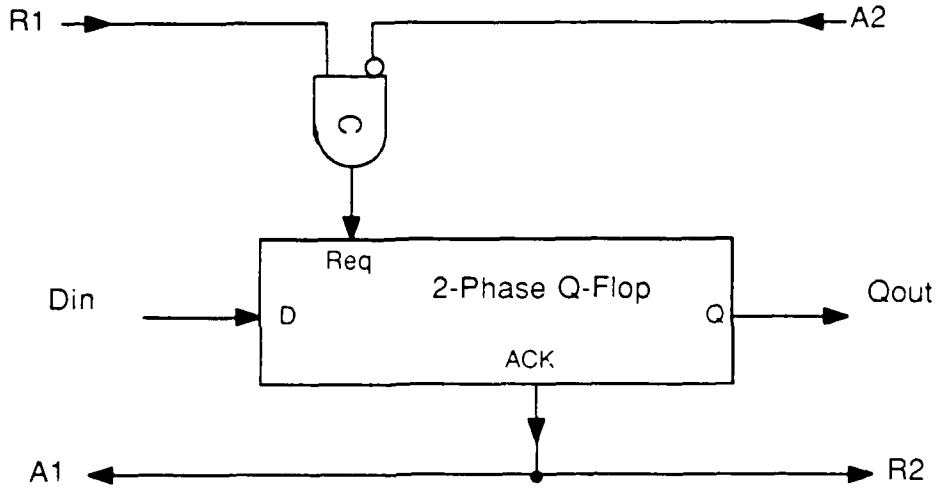


Figure 34: A FIFO Cell Using A Two Phase Q-Flop

Configuration

There are two configurations on Parts-R-Us that consist of different types of Q-flops. The four phase Q-flop configuration includes two Q-registers. These are collections of Q-flops with common request inputs and acknowledgments combined in a C-element so that the register will acknowledge only after every bit in the register acknowledges. This configuration, shown in Figure 35, includes an eight bit wide Q-register and a five bit wide Q-register, as well as a single four phase Q-flop. The five bit wide Q-register includes a sampling feature similar to that in the mutual exclusion element from section 2.5.. For normal operation of that Q-register, the sample input should be tied high.

The two phase Q-flop configuration of Parts-R-Us includes two two phase Q-registers; one eight bits wide and the other seven bits wide. This is shown in Figure 36. As in the four phase case the input requests are shared and the acknowledgments are combined into a C-element.

3. Design and Testing

Parts-R-Us was designed during spring semester 1986. The layout was done using the Magic layout program from U.C. Berkeley. The chip was also specified using the Net circuit description language. The Net description was simulated with RNL, and the extracted netlist from the layout was compared to the Net-generated netlist with Gemini. After this comparison it is not hard to believe that the circuit that correctly ran the simulations is actually the same as the circuit drawn in the layout. Parts-R-Us was designed using the Scalable CMOS design rules and was fabricated through the MOSIS on run M67Y at lambda = 1.5 microns in August 1986 which results in a three micron feature size. The chip uses a MOSIS standard 6800x6900 micron 40 pin pad frame.

Packaged chips arrived in late August 1986 and were test·i using a simple functional tester designed that same summer [3,1]. The test programs for the simulation are written in CommonLISP

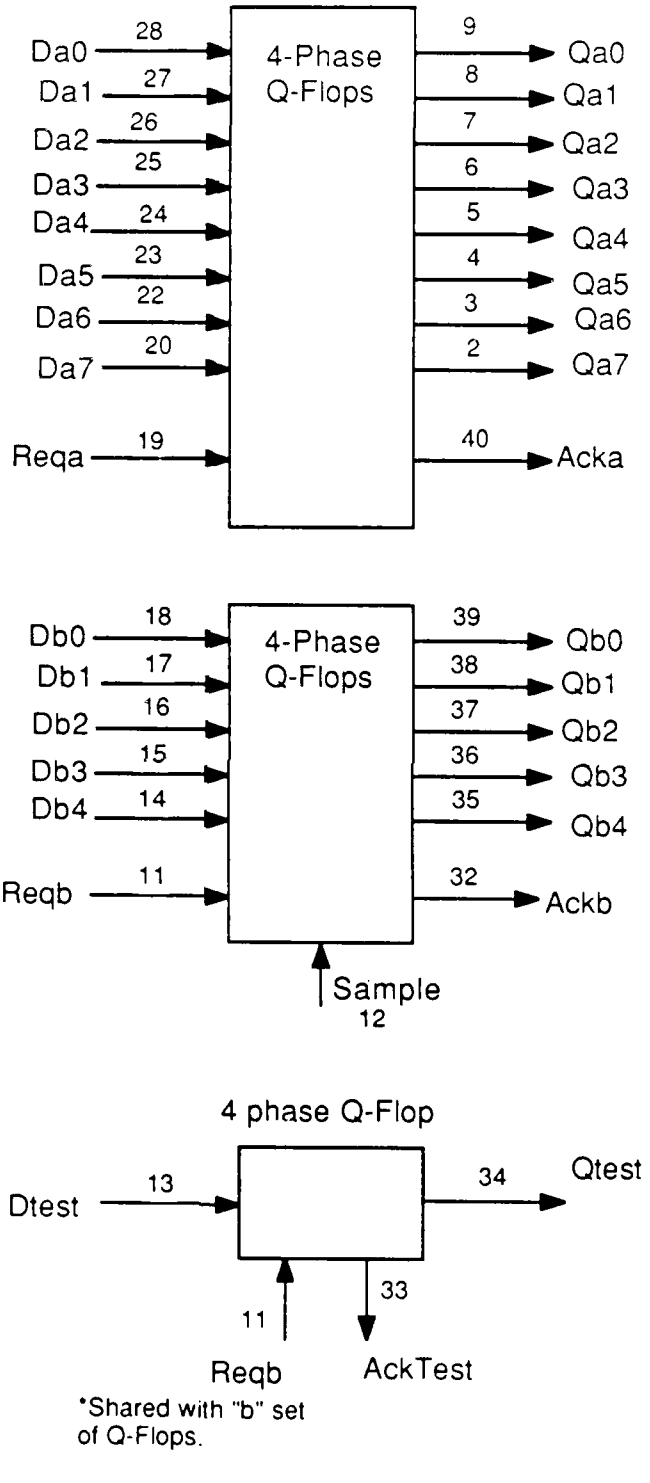


Figure 35: Four Phase Q-Flop Configuration

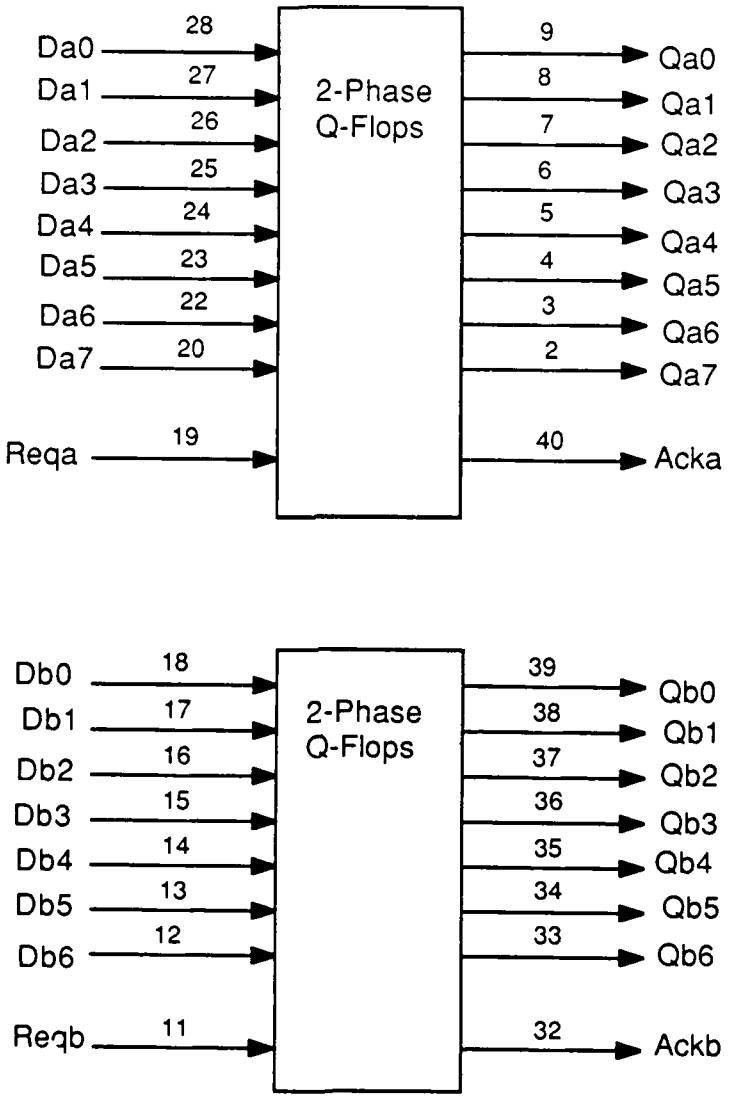


Figure 36: Two Phase Q-Flop Configuration

using a LISP package designed to describe tests. This package can drive both the RNL simulator and the simple tester so that the exact same test program may be run on both. This procedure makes initial testing of the chips when they come back very easy since the test programs have already been written.

While the functional tests of the packaged parts were very encouraging, every type of part included on Parts-R-Us was functional somewhere, the yield was a little disappointing. Of the 13 packaged parts tested, two were very close to completely functional, two were completely non-functional, and the rest fell somewhere in between. A complete table of testing results can be seen in Figure 37.

In addition to functional tests, some tests of the metastable properties of the Arbiters and Q-flops were conducted. In the case of the two-way Arbiters, the two competing requests were driven at very close to the same time into the Arbiter. One of the requests was driven through a constant-impedance trombone-style variable delay line so that the relative arrival times of the two requests could be carefully controlled. As the variable delay was adjusted the arbiter would decide in favor of different requests and right at the decision point the grant signal would take longer to resolve as the internal mutual exclusion element resolved from its metastable state.

For the Q-flops, the data and request inputs were driven through the same sort of system. When the data and the request to sampie happen very close to the same time, the acknowledge can be seen on the oscilloscope to take longer to arrive. For the Q-registers, this procedure repeated for each bit of the register can verify that the large C-element that combines the acknowledgments of each bit is actually operating correctly. That is, the delay slowly changing on each bit should make the register acknowledgment take longer for some value of the delay.

Since it was possible to use the metastable test rig to get information about the metastable exit properties of the circuits, the sampling feature included on the mutual exclusion element and four phase Q-flops was not used at all. Parts-R-Us does not contain any other on-chip circuitry for testing. While the test sequences used for the functional test were rather ad hoc, the individual modules on Parts-R-Us are small and don't really contain much internal state that would be useful for testing. Most of the state of the modules is visible from the outputs. A more careful testing strategy in which all internal nodes of a module are exercised should be done but the completed functional tests are still fairly convincing. Since there is some extra space on the chip, future versions of Parts-R-Us could also include self-test circuitry.

4. Conclusions

While the yield was not great for this run of Parts-R-Us, the fact that every part worked on at least one of the packaged parts indicates that the design is probably not at fault. There are enough parts available on the 13 chips to build small circuits, although only one circuit has been built to date. A circuit that uses the C-types from the C-type configuration to build a one bit wide, eight bit deep four wire FIFO was built in December 1986.

Parts-R-Us has demonstrated that the asynchronous building blocks it contains actually work and can be used to build small asynchronous control circuits. The packaged chips are available to interested persons to use for prototyping circuits.

Individual Chip Results

Chip #	Ctype ccccddffff 12345121234	Call ccccc 12345	Toggle ttttccccc 123451234	Select ssssss 123456	FIFO fccc 1123	4pQ qqq 851	2pQ qq 87	Arbit aamcc 12112	% OK
1	xxxxxxxxxxx	x	xxxxxxx	xx x	xx	x x			57%
2	x x xxxxxx	x	xxxx x	x	xxx	xx		xx	40%
3	x xx xxxxxx	xxx	xx x xx	xx x	x	x		x xx	55%
4	x xx xxx	xxxxx		xxxxxx		x			40%
5	xxxxx x xx	xxx	x xxxx	xxx		x		x	46%
6	x x xxxxx	x x	xx xx x	x	xx	x	x	x x	46%
7	xx xxxxxxxx	xx x	xxxxxxx x	xxxxxx	xx x	x	x	x	73%
8	xxxxxxxxxx	xxxxx	xxxxxxxxx	xxxxxx	xxxx	xxx	x	xxxxx	98%
9									0%
10	xxxxxxxx		xxx	x xx	xxx	x		xx	44%
11	xxxxxxxxxx	xxxxx	xxxxxxxxx	xxxxxx	xxxx	xxx	x	xxxxx	98%
12	x x x x	x	xxx x	xx				x	27%
13									0%

Legend: x = The part passed its functional tests

Overall Chip Results

Configuration	Part Name	Total Possible	Total Working	% Good
C-type	C-type	52	28	54%
C-type	xor	78	54	69%
Call	2-way call	52	23	44%
Call	4-way call	13	5	38%
Toggle	toggle	65	33	51%
Toggle	C-type	35	17	49%
Select	2-way select	65	22	34%
Select	4-way select	13	9	69%
FIFO	register	13	3	23%
FIFO	C-element	39	19	49%
4-Phase Q	8-wide Q-reg	13	5	38%
4-Phase Q	5-wide Q-reg	13	5	38%
4-Phase Q	Q-flop	13	6	46%
2-Phase Q	8-wide Q-reg	13	4	31%
2-Phase Q	7-wide Q-reg	13	0	0%
Arbiter	arbiter	26	9	35%
Arbiter	mutex	13	4	31%
Arbiter	2-way call	26	9	35%

Figure 37: Test Results of the 8/86 Version of Parts-R-Us

BIBLIOGRAPHY

[1] Erik Brunvand and Ivan Sutherland. *An Asynchronous Q-Bus Interface.* Technical Report 4677, Sutherland, Sproull and Associates, 1986.

[2] F.U. Rosenberger, C.E. Molnar, T.J. Chaney, and T.P. Fang. *Q-Modules: Internally Clocked Delay-Insensitive Modules.* Technical Report, Washington University, St. Louis, 1985. To appear in IEEE Transactions on Computers.

[3] Bob Sproull and Ivan Sutherland. *A Simple Tester.* Technical Report 4676, Sutherland, Sproull and Associates, 1986.

[4] Robert F. Sproull and Ivan E. Sutherland. *Asynchronous Systems.* Technical Report 4706, 4707, 4708, Sutherland, Sproull and Associates, 1986.

[5] Robert F. Sproull, Ivan E. Sutherland, Charles E. Molnar, and Edward H. Frank. *Asynchronous Systems.* Technical Report 3441, Sutherland, Sproull and Associates, 1985.

[6] Ivan Sutherland, Bob Sproull, and Ian Jones. *Standard Asynchronous Modules.* Technical Report 4662, Sutherland, Sproull and Associates, 1986.

END
DATE
FILMD
3-88
DTIC